Quantities in Games and Modal Transition Systems

Juhl, Line

Publication date:
2013

Document Version
Accepted author manuscript, peer reviewed version

PhD Dissertation

Quantities in Games and Modal Transition Systems

Line Juhl

Aalborg University
Department of Computer Science

Abstract

As embedded software becomes an increasing part of our daily lives, modelling and verification of embedded systems is highly relevant. Common for many embedded software systems is the restricted use of and access to resources. Thus modelling and verification of embedded systems must not only rely on functional requirements, but also extra-functional requirements such as fuel consumption or bandwidth usage. This thesis presents formalisms for modelling such quantitative systems and focuses on both the inherent reactive characteristic of embedded systems, implied by the surrounding environment, as well as on the need for compositional reasoning.

The thesis consists of two parts. The first part is an introduction that motivates the need for formal methods, and model checking in particular, when developing reactive systems with quantitative resource constraints. Then the formalisms for weighted reasoning are defined, and the challenges of embedded systems are discussed in order to identify useful methods and theories for dealing with the quantitative aspects of reactive systems. This includes game theory for modelling the reactive aspects and modal transition system as the basis for a compositional specification theory allowing a stepwise refinement.

The second part of the thesis contains five papers. The first two papers are on the recently emerged notion of energy games, where we study the existence of infinite runs in multiweighed games subject to boundary constraints on the accumulated weights. The boundary constraints considered are both lower and upper bounds. We give tight complexity bounds for deciding the problem in a number of different settings and show undecidability in two other cases. In the case of unknown upper bounds we provide a method for constructing the exact set of upper bounds yielding such infinite runs. The remaining three papers study and define weighted modal transition systems. We present an algorithm running in polynomial space for finding the most general common refinement for deterministic specifications and a sound quantitative extension of CTL reasoning on weighted modal transition systems. Two extensions to a multiweighed setting are given in the two succeeding papers. The first extension allows for refinable labels and we prove that the key notions of modal and thorough refinement and determinisation behave as expected as well as the operators for structural and logical composition and the quotient operator. The second extension presents a logic for reasoning on the accumulated weight along the runs. The semantics of the logic is game-based and is thereby proven to be both sound and complete. The logic is proven undecidable in general, however useful decidable fragments are identified.

Dansk sammenfatning

Da brugen af indlejret software i vores dagligdag bliver mere og mere udbredt, er modellering og verifikation af indlejrede softwaresystemer særdeles relevant. Fælles for mange indlejrede systemer er den begrænsede brug af og adgang til ressourcer. Derfor må modellering og verifikation af indlejrede systemer ikke kun bero på rent funktionelle egenskaber, men også kvantitative egenskaber så som energi- eller båndbreddeforbrug. Denne afhandling præsenterer formalismer til modellering af sådanne kvantitative systemer og fokuserer på både den reaktive adfærd ved indlejrede systemer, såvel som behovet for at analysere sammensatte systemer.

Afhandlingen består af to dele. Den første del er en introduktion, der begrunder brugen af formelle metoder og model checking i særdeleshed, når der udvikles reaktive systemer, hvor brugen af kvantitative ressourcer restringeres. Efterfølgende defineres formalismer, som er brugbare til analyse af kvantitative systemer, og egenskaber ved indlejrede systemer identificeres, således at brugbare metoder og teorier til håndtering af de kvantitative aspekter ved reaktive systemer kan identificeres. Dette inkluderer spilteori til at modellere de reaktive aspekter samt modale transitionssystemer som en basis for en specifikationsteori for sammensatte systemer, der tillader trinvis forfining.

Anden del af afhandlingen indeholder fem artikler. De første to artikler omhandler en nyligt opstået type af spil kaldet 'energispil', hvor vi behandler eksistensen af uendelige stier i multivægtede spil, hvor den samlede akkumulerede vægt langs stien er underlagt øvre og nedre grænser. Vi giver matchende øvre og nedre grænser for kompleksiteten af dette afgørbarhedsproblem i en række forskellige varianter af energispil samt viser uafgørbarhed i yderligere to tilfælde. I det tilfælde, at den øvre grænse er ukendt, giver vi en metode til at konstruere den præcise mængde af øvre grænser, der giver anledning til eksistensen af sådanne uendelige stier med begrænset ressourceforbrug. De resterende tre artikler definerer og analyserer vægtede modale transitionssystemer. Vi præsenterer en algoritme, der bruger polynomiel plads, som finder den mest generelle fælles forfining af en række deterministiske systemer og en sund kvantitativ udvidelse af logikken CTL, der behandler vægtede modale transitionssystemer. To udvidelser til en multivægtet udgave gives i de to efterfølgende artikler. Den første udvidelse tillader, at labels forfines, og vi viser, at de vigtige begreber modal og semantisk forfining opfører sig som ventet. Det samme er tilfældet for operatorerne for struktural og logisk sammensætning samt kvotientoperatoren. Den anden udvidelse præsenterer en logik til analyse af den akkumulerede vægt langs stierne. Semantikken for denne logik baseres på spilteori, og vi beviser derved, at den er både sund og fuldstændig. Yderligere bevises det, at logikken er uafgørbar i det generelle tilfælde, og der identificeres anvendelige dele af logikken, der er afgørbare.

Acknowledgements

My sincere gratitude goes to my supervisor, Kim G. Larsen. Thank you for always being supportive and positive and for your inexhaustible source of good ideas and helpful comments. I am grateful that you kept believing that it is possible to turn a mathematician into a computer scientist.

An equally big thank you goes to my co-supervisor, Jiří Srba, who was so kind to take me under his wing as a PhD student. Your immense help regarding every aspect of this thesis, from how to structure an introduction to how to keep a deadline has been invaluable.

I would also like to thank my office mates, Mikkel Larsen Pedersen and Claus Thrane. Thank you for making work a joy (almost) every day and for answering all my naive questions.

During my work I had the pleasure of spending two months in Bruxelles at Université Libre de Bruxelles. I would like to thank Jean-François Raskin for being a great host and for providing me with new ideas and insight. And a huge thank you to my office mate at ULB, Mahsa Shirmohammadi, who welcomed me with such open arms that it brought our two directions of research together.

Without my additional co-authors, Sebastian S. Bauer, Axel Legay and Uli Fahrenberg, the papers included in this thesis would not exist, so I am sending a big thank you to them as well.

Last I would like to thank my mum for her endless support, my dad for teaching me to be just like him, and my little brother for loving me.

Line Juhl
February 2012

Contents

I. Introduction 1

1. Motivation 2
- 1.1. Model Checking . . . 3
- 1.2. Resources . . . 4
- 1.3. Synthesis . . . 6
- 1.4. Formal Specification . . . 7
- 1.5. Research Objectives . . . 8

2. How to Model Resources 10
- 2.1. Modelling Phase . . . 10
  - 2.1.1. Transition Systems . . . 10
  - 2.1.2. Composing Systems . . . 12
  - 2.1.3. Modelling Resources . . . 13
  - 2.1.4. Composing Weighted Systems . . . 14
- 2.2. Specification Phase . . . 15
  - 2.2.1. Logical Formalisms . . . 15
  - 2.2.2. Quantitative Requirements . . . 17
- 2.3. Multiple Quantities . . . 19

3. Adapting to the Challenges of Embedded Systems 22
- 3.1. Games . . . 22
- 3.2. Modal Transition Systems . . . 25
  - 3.2.1. Refinable Sets of Labels . . . 27
  - 3.2.2. Logical Characterisation . . . 30
  - 3.2.3. Completeness Using Games . . . 33
- 3.3. Metrics . . . 34

4. Thesis Summary 37

II. Papers 43

A. Energy Games in Multiweighted Automata 44
- 1. Introduction . . . 45
- 2. Multiweighted Automata and Games . . . 47

3. Relationship to Petri Nets . . . 49
4. Reductions among Energy Games . . . 51
5. Summary of Complexity Results . . . 56
6. Parameterized Existential Problems . . . 58
7. Extension to Timed Automata . . . 60
8. Conclusion and Future Work . . . 63

B. Optimal Bounds for Multiweighted and Parametrised Energy Games 64
1. Introduction . . . 65
2. Multiweighted Energy Games . . . 67
3. Weak Upper Bound . . . 68
4. Strict Upper Bound . . . 70
5. Parametrised Transitions . . . 72
6. Conclusion and Future Work . . . 73

C. Modal Transition Systems with Weight Intervals 75
1. Introduction . . . 76
2. Definitions . . . 79
3. Largest Common Refinement . . . 83
4. Logical Characterisation . . . 88
5. Conclusion and Future Work . . . 95

D. Extending Modal Transition Systems with Structured Labels 97
1. Introduction . . . 98
2. Label-Structured Modal Transition Systems . . . 101
3. Specification Theory . . . 113
4. Logical Characterization . . . 125
5. Conclusion . . . 129

E. A Logic for Accumulated-Weight Reasoning on Multiweighted Modal Automata 130
1. Introduction . . . 131
2. Multiweighted Modal Automata . . . 132
3. Games on Multiweighted Modal Automata and Logic $\mathcal{L}$ . . . 134
4. Decidability and Complexity of the Logic $\mathcal{L}$ . . . 138
5. Conclusion . . . 148

Bibliography 148

Part I.

Introduction

1 Motivation

For decades software systems have been synonymous with desktop computers or, in the later years, laptops. However, nowadays we are surrounded by software systems everywhere in life. This be in our cars, our phones or our TVs at home. Such systems are known as *embedded systems*, and are special-purpose systems built to handle a specific task for the electronics that it is part of. Due to the very nature of embedded systems, they often react to stimuli from the surrounding environment and are usually subject to a hardware platform with limited resources. Embedded systems are thus also characterised as engineering artifacts involving computations subject to physical constraints [HS06].

As any other piece of software, embedded systems can be subject to errors. Such errors are of course highly undesirable in safety-critical systems such as aeroplanes or cars, but also for economical reasons surely no company is interested in errors in their products if they can be avoided at a limited cost. Since we are increasingly dependent on embedded systems (both safety-critical and non-safety-critical) in our daily lives, it is of uttermost importance that the number of errors appearing in such systems are kept to a minimum. Furthermore the systems developed nowadays are in general significantly larger and more complex than before, thus making it increasingly hard and time-consuming to develop error-free software. Reducing the time spend on developing a software systems without lowering the quality of the product is therefore of huge interest to any company.

Ensuring the correctness of a piece of software is a highly non-trivial task. One way is to use *formal methods*, which use mathematically-based techniques to identify or exclude errors. Formal methods allow a design and its desirable requirements to be described in a unambiguous manner such that one by mathematical reasoning can prove or disprove the correctness of the model with respect to the expressed requirements—so-called formal verification. Given the appropriate tool support, this approach can ensure a more reliable system. Furthermore, a formal description can help designers, programmers and end users to avoid misunderstandings, as they will have a common agreement on the behaviour of the system. Formal methods can be deployed at many

stages in the design process and errors can be found early in the process, reducing the cost of correction greatly [Pel01, BK08, AVARB+01].

Formal methods are increasingly used in the software industry to verify software systems [TWC01, HLP01], yet many companies still avoid formal methods due to several different factors. Firstly, formal methods appear time and resource consuming, which often conflicts with tight deadlines and budgets. The fact that formal methods in the long run may save both time and money seems to be neglected. Secondly, the learning curve for some formal methods is somewhat steep and as a consequence might discourage potential new users.

1.1. Model Checking

Over the last 30 years an entire zoo of formal methods for verifying software systems have been developed, each with different strengths and weaknesses. The contributions of this thesis are within the formal verification method called *model checking*, which has been successfully utilised for several industrial cases [CAP+98, SAH+00, Low96, Ern05, JRLD07]. Model checking is a verification technique that given a finite mathematical model of a system, attempts to logically prove that the model satisfies the properties given by some formal specification. The specification must also be given within some suitable mathematical framework, most often *modal* and *temporal logics*. The initial work on model checking was done independently in the early 80s by Clarke and Emerson [EC80, CE82] and Queille and Sifakis [QS82]. Clarke, Emerson and Sifakis were all awarded the Turing Award in 2007 for their pioneering work on model checking.

In order to perform the actual verification of the model, algorithms automating the verification process are useful. Such algorithms will often make an exhaustive search of the entire model if necessary in order to verify a given property. Unfortunately, model checking suffers from the *state space explosion* problem, entailing that the number of states needed in the model grows exponentially with the number of submodels of the model. This increased computational complexity makes in some cases model checking infeasible. However, the extensive research in the area over the years has contributed to the development of faster algorithms and more efficient data structures for exploring and representing state spaces. Combined with new and faster computers, model checking is nowadays not only restricted to toy examples, but can actually be applied to large-scale industrial designs and is therefore widely accepted as a useful verification method today [BK08]. The strengths of model checking lie in the sound mathematical framework and the exhaustive (and automatic) search through the entire model. This approach logically proves whether or not the model satisfies a given property and as a consequence the developers can trust the result of the verification—given that the tool analysing the model is flawless and that the model and the specification reflects the intended design and requirements. The model checking approach is contrary to the well-known (and widely used in industry) validation techniques of testing or simulation, where the complete model may not be explored, thus not necessarily revealing an error even though searching for it [Mye79]. An additional strength of model checking is that a negative

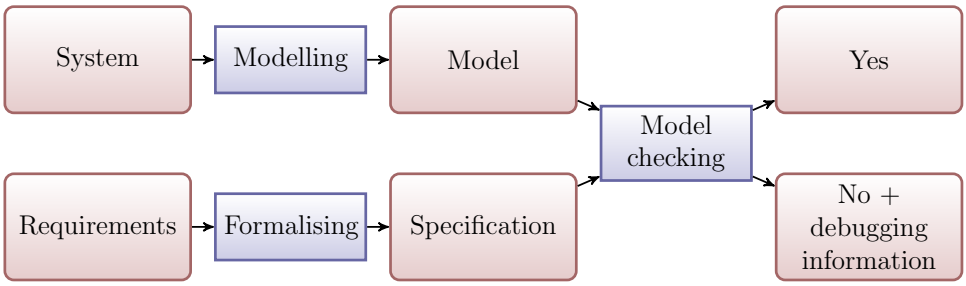


Figure 1.1.: Model checking

verification result can provide debugging information, helping to locate the error.

In summary model checking can be divided into the following three phases; modelling, specification, and verification phase. Figure 1.1 depicts the three phases in connection. Firstly, the concrete system in mind is modelled using some suitable formalism. Secondly, the requirements that the system should adhere to are formalised in some mathematical specification language. The model and the specification are then fed to the model checker that answers either yes, the model satisfies the specification, or no, the model does not. In case of a no, some debugging information can be provided, helping the user modify the system or the specification in order to achieve a positive result. Being able to precisely model the design and specify the properties in mind is a crucial point to model checking, since imprecise modelling can lead to imprecise results. This is also one of the drawbacks of model-based verification; surely the verification is only as good as the model of the system, since we do not verify the code itself, but merely a model of the code. It is therefore important that we develop formalisms which are capable of expressing all the aspects of a given design that are needed for a correct verification process. Another drawback is that only the stated properties bare verified. This gives rise to unexpected errors that remain undiscovered. Thus also being able to formulate the right properties is crucial to ensure correctness.

Due to the increasing use of embedded system (and thus also the demand for correctness of such), we seek to identify some of the challenges they bring to the different phases of model checking. The hope is to enhance model checking and make it more suitable for verification of embedded systems. These challenges include explicit handling of both the quantitative aspects and the inherent reactive nature of embedded systems as we will discuss in the following sections.

1.2. Resources

Correctness of software systems can be phrased in many ways. For a piece of software running on a laptop, we might want to establish correctness as a certain functionality that the program must posses. This could be the absence of a deadlock or that some request-response requirement is always fulfilled. Such requirements address the services ensured for the end user and are called *functional requirements*.

For modern embedded systems, such functional properties might not be sufficient

to adequately express the requirements of the system defined by the customer. Since embedded systems are highly dependent on their actual hardware platform, we need to construct properties dealing with the limited resources available for certain systems. For instance the specification of an airbag for a car should not only require that the airbag is inflated in case of an accident, but also insist that this behaviour is executed within some strict time delay following the accident. An environmentally friendly coffee machine might have the requirement that all cups of coffee should be served with some maximum consumption of energy. These properties addressing the quality of the services and involving the resources available on the execution platform are also known as *extra-functional requirements* [HS06].

The resources of a system may be divided into boolean or quantitative resources. A boolean resource could be a meeting room at a university or a designated part of the memory on a computer. These resources are characterised as resources that a system either holds or does not hold. In case the system (or process) holds the resource, no other system can hold it simultaneously. Semaphores are often used in programming when dealing with boolean resources. A quantitative resource is for instance energy, bandwidth, time, volume or temperature. When obtaining these resources we use a numerical value to measure the amount, as we always obtain some quantity of the resource. The use of a such resource is also independent of the use of the same resource by other systems. This thesis will solely concentrate on quantitative resources, and thus in the remaining thesis a resource always refers to a quantitative resource.

The need for extra-functional requirements and therefore also explicit handling of quantitative resources has been around for many years and as a result much work has been done in this area already. Both formalisms for implementations and specifications have been equipped with quantitative information. This includes both logics, process algebra and graphical formalisms. We will present examples of such formalisms in Chapter 2. Often an (mathematically) abstract approach is chosen when modelling quantities, since they then can be applied to a broader range of problems. Still, a too general model may also be useless for practical purposes due to its opaqueness and possible computational limitations.

An even more general (but highly relevant) approach is to consider multiple quantities in the same model or specification. In real life it is usually not the case that an embedded system only reacts to precisely one quantitative stimuli. Consider an automatic robot built for an expedition to Mars. Such a vehicle will not only have to reason about its own power supply, but likely also temperature, pressure, speed, etc. Moreover the robot may enter a situation where a trade-off between these monitored variables is necessary—should I keep the power consumption low or should I move at a higher speed in order to survey more ground in less time? As evident, allowing multiple quantitative resources introduces new considerations and new properties to be explored.

Another feature of many embedded systems compared to traditional software systems running on desktop computers is their non-terminating nature. The majority of traditional systems perform some service once and then terminate, while this approach is unsuitable for many embedded system. A traffic light or an elevator (or even an operating system) must keep functioning "forever" and have no termination criteria. The need

for quantitative reasoning for such infinite behaviour is thus highly relevant.

1.3. Synthesis

As described previously, a characteristic of many systems nowadays is their reactive nature. Systems are usually subject to some sort of communication with either other systems, the environment or a user. A *reactive system* is a system that computes by reacting to stimuli from the surrounding environment [HP85]. Many embedded systems are reactive, since many read the value of some sensor (e.g. temperature, velocity) and then conduct some computations depending on the concrete value and take actions by setting actuators. Seen from the system's point of view, some actions are controllable by the system while the ones performed by the environment are uncontrollable. This could for instance be the case for a conveyor belt, as it is in charge of adjusting its own speed according to the uncontrollable flow of items put onto the belt. Another example would be a thermostat adjusting the temperature. Here the thermostat is not in control of environmental changes that effect the temperature, but can adjust the thermostat based on the readings of the current temperature.

As a developer of such a reactive system, it is often useful to investigate whether your system can behave in such a way that no matter how the environment behaves, some requirement is still satisfied. In the positive case one is interested in constructing a controller doing the actual task of responding to the environment in a suitable way. Such a controller cannot be constructed using model checking, as this only yields 'yes' information in the positive case. The task of constructing a controller reacting with the environment in a satisfactory way is called *synthesis*. An overview of the tasks of synthesis can be seen in Figure 1.2. Synthesis is somewhat similar to model checking, as we both need a specification and a model of the system as input. However, the system is given as two subsystems. One system represents the uncontrollable environment while the other represents the controllable system interacting with the environment. The outcome of the synthesis is a concrete strategy in both cases. In the positive case the strategy will provide the user with information on how to perform the controllable actions in order to achieve the required property. In the negative case, the strategy will provide information on how the environment can perform the uncontrollable cases to prevent that the requirement is satisfied.

When dealing with systems having both controllable and uncontrollable actions, the mathematical concept of *game theory* is often used when creating a model of the system. In general, game theory is used when describing situations where two or more individual agents (or *players*) each make a series of decisions. An outcome of the game for each player is determined on the basis of all the decisions made by the players. Each player is interested in maximising his own outcome, and he therefore seeks to make his decisions in a way that makes his outcome best possible, no matter the decisions of the opponents. As the description indicates, the game theoretic framework has a long list of applications, ranging from "simple" card games such as poker [MS07] to complex economics [vNM44] and biology [MP73].

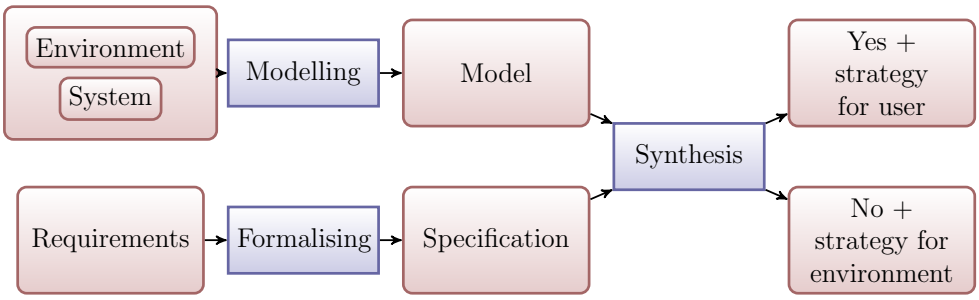


Figure 1.2.: Synthesis

1.4. Formal Specification

We now turn to the specification phase of model checking. In order to express the requirements that the system should satisfy, a formalism for doing so is required. Below we discuss what features a suitable specification formalism should encompass.

The increasing complexity in embedded software is due to both an increasing size of the systems, but also a more interactive nature of nowadays systems. Such systems may for sure also react with each other, thus a large system may be described by a number of smaller reactive systems, increasing the complexity of the entire system considerably.

This growing complexity entails that it is in practice very hard to reason about the correctness of such a system, no matter the method. In order to accommodate this problem, large software systems are divided into smaller and more transparent components (as described using reactive systems above), which are designed by independent teams. This approach may at first glance further complicate the verification process, since not only must each component be correct, the components must also interact with each other in the correct way. By defining an interface for each component that describes how the component should react to the outside world, the interplay between components can be ensured. The job of the individual developer teams is to design and implement a specific component such that it complies with a given interface. In the view of formal methods, these interfaces can be seen as specifications, while the specific components can be seen as the concrete model or the implementation. Thus we may reason about systems on a more abstract level by letting a specification act as an abstraction of a set of implementations. It is for these reasons natural to develop formalisms for reasoning at the specification level that allow independent reasoning on each component such that a simplified design can be achieved through compositional reasoning.

Any such reasonable *specification theory* should be equipped with both a satisfaction and a refinement relation to compare an implementation with a specification and to compare two specifications, respectively. The last relation allows for *stepwise refinement* such that a gradual refinement process is possible. In this way we may start out with a general specification and by a series of refinements restrict the specification further before finally ending up with a concrete implementation. Furthermore operators for logical and structural composition of systems should be present. Logical composition is useful when searching for a common implementation given several specifications that

must be fulfilled. Structural (or parallel) composition is needed due to the inherently parallel nature of reactive systems. The dual to structural composition, the quotient operator, is also an important ingredient. This operator constructs a sub-specification given an overall specification and another sub-specification refining a part of the overall specification.

Usually two approaches can be taken when constructing a specification formalism. A *logical* and a *behavioural* one. The logical approach entails that the specification is given using some logic and the model checking is done by investigating the denotational semantics of the logic. Appropriate logics will be mentioned in Section 2.2.1. Logics are very useful when reasoning on the logical composition of specifications and can also provide stepwise refinement along with the usual notion of a model satisfying a specification given by a logical formula. However, logics are not suitable when addressing the structural composition of systems, as the structural composition of two logical formulae is so far not so well-understood [GS86, Hol89, LX91, ASW94].

Similarly, a *behavioural* approach can also be taken to express the specification using a formalism similar to the one used to model the system (the implementation). The verification procedure is then done by comparing the behaviour of the two systems using for instance equivalence relations or preorders. Formalisms for this approach could be the graphical formalism of transition systems [Kel76] or the one of *process algebras*, e.g. Hoare's CSP [Hoa85], Milner's CCS [Mil80] or ACP by Bergstra and Klop [BK84]. Process algebras are suitable for parallel composition, but when considering the logical composition, process algebras fall short. The constructed logical composition will either be empty or be bisimilar to both of the components. This operator is thus not of much use when studying specifications as process algebras.

Another formalism accommodating the behavioural approach is the formalism of *modal transition systems* (MTS) formulated by Larsen and Thomsen in [LT88a]. This formalism is capable of specifying optional behaviour, making the formalism "loose", as it can specify several non-bisimilar implementations. A modal transition system therefore gives rise to a set of (non-bisimilar) implementations in the same way as a logical formula is satisfied by a number of transition systems. This formalism both ensures compositional reasoning and stepwise refinement, being two important features when addressing embedded systems.

1.5. Research Objectives

The preceding sections have argued why errors in embedded software are highly undesirable and why verification through formal model-based methods is a recommendable approach in order to keep the number of errors and the time spend on finding them to a minimum.

The aim of this work is to further strengthen the model checking framework available for embedded system by tailoring the formalisms to the challenges of embedded systems. In the above it has been argued that models and specifications reasoning on quantities are needed when addressing such systems. Moreover, games and component-based specifica-

tion theories should be utilised in order to fully capture the reactive nature of nowadays computer systems. This work intends to enhance the model-based theory of embedded systems to incorporate quantitative aspects.

This will be done by considering the following sub-questions, which can be grouped under three general headlines.

1. Quantitative resources and specification formalisms
   i) How can quantities be modelled using transition systems, logics and modal transition systems?
   ii) What does a logic that reasons on quantitative modal transition systems look like?
2. Quantitative resources and games
   iii) What can be said about the use of limited resources when assuming an infinite behaviour in games?
   iv) Can game theory be applied to logics for modal transition systems?
3. Multiple resources
   v) What is the computational cost of the above problems when assuming multiple resources?
   vi) How to identify the optimal usage of multiple resources when assuming an infinite behaviour?

2 How to Model Resources

As advocated in the previous chapter, many systems, especially embedded systems, are highly resource dependent. Therefore we are interested in extending model checking for asking and verifying properties that reason on resources.

This section proposes a suitable way to incorporate quantities into both the model and the specification. When choosing a model for modelling a hardware or software system, it is important that the chosen model has the right level of abstraction. The model must capture all details necessary to check the property in mind, but on the other hand not capture features of the system which are unnecessary for the property.

The choice of modelling formalism is thus of great importance. The same applies to the specification formalism, which should of course be able to state the property in mind—in our case properties related to the resource constraints or requirements of the system.

We start by adding quantities to the modelling formalism of *transition systems* and then move on to the question of quantitative properties. Finally we shall address the notion of multiple quantitative aspects.

2.1. Modelling Phase

In the following we present the well-known formalism of transition systems and examine the quantitative aspects that we seek to incorporate in order to obtain a simple model for quantitative systems.

2.1.1. Transition Systems

One of the most used modelling formalisms for model checking is the one of transition systems. The formalism was first employed for verification by Keller in [Kel76] and later by Plotkin for structural operational semantics in [Plo81]. A transition system describes

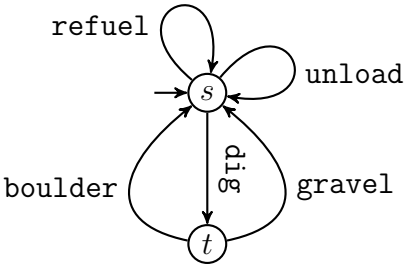


Figure 2.1.: An excavator modelled as a transition system

a system as consisting of a number of states, each state being a snapshot of the system at a given time. The behaviour of the system is modelled using transitions, which relate pairs of states. A transition can also be seen as some form of "handshake" with the given system and its environment. The environment may be another (or several other) system(s) or an external user. It is common to decorate transitions with labels to denote the type of behaviour that causes the state change, making the model more descriptive.

Definition 1 (Transition system)**.** A transition system is a tuple $T = (S, L, s_0, \longrightarrow)$, where S is a set of states, L is a set of labels, $s_0 \in S$ is the start state and $\longrightarrow \subseteq S \times L \times S$ is a transition relation.

Thus a transition system is essentially a directed graph with some information assigned to each edge and a designated initial state. The above definition may also be referred to as a *labelled transition system*. For convenience we write $s \xrightarrow{a} t$ if $(s, a, t) \in \longrightarrow$.

An *execution* (or *run*) π of a transition system is a series of states and transitions starting with the initial state, such that $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots$. A run can be either finite or infinite and denotes one possible outcome when executing the actual system. A *maximal* run is either infinite or a finite run that ends in a state with no outgoing transitions. We denote the set of maximal runs starting from s_0 as $\mathsf{runs}(s_0)$.

Example 2. As an example of a transition system, consult Figure 2.1. The transition system models an excavator. The start state s is indicated by the arrow with no origin. The excavator may here choose to `dig`, `refuel` its tank or `unload` its dipper. If the machine `dig`s, it can choose between picking up `gravel` or a `boulder`. A possible finite execution of the excavator is $s \xrightarrow{\texttt{dig}} t \xrightarrow{\texttt{boulder}} s \xrightarrow{\texttt{unload}} s \xrightarrow{\texttt{refuel}} s$.

Notice that a transition system can model nondeterminism, since one state may have more than one outgoing transition assigned the same action. If on the other hand $s \xrightarrow{a} t$ and $s \xrightarrow{a} t'$ implies $t' = t''$ for all $s, t, t' \in S$ we call the system *deterministic*.

$$\text{(A1)}\ \frac{s_1 \xrightarrow{a}_1 t_1}{s_1 \parallel s_2 \xrightarrow{a} t_1 \parallel s_2} \qquad \text{(A2)}\ \frac{s_2 \xrightarrow{a}_2 t_2}{s_1 \parallel s_2 \xrightarrow{a} s_1 \parallel t_2}$$

Table 2.1.: Interleaving semantics

$$\text{(A3)}\ \frac{s_1 \xrightarrow{a}_1 t_1 \qquad s_2 \xrightarrow{a}_2 t_2}{s_1 \parallel s_2 \xrightarrow{a} t_1 \parallel t_2}$$

Table 2.2.: Synchronising semantics

2.1.2. Composing Systems

An important aspect of all software systems, and embedded systems in particular, is that of concurrency, meaning that different behavioural aspects are under the control of designated components all operating simultaneously interacting between components. It is therefore important that the chosen modelling formalism is capable of representing concurrency in a suitable way. The modelling formalism should have a composition operator $\parallel$ that, given a number of systems $T_1, \ldots, T_n$ running simultaneously, constructs the full system, denoted $T_1 \parallel \ldots \parallel T_n$.

For transition systems the operator mentioned above can be defined in several ways. One approach to modelling concurrency is using a purely interleaving view, where each subsystem can perform its actions independent of the other subsystems, allowing the resulting transition system to interleave the transitions of the subsystems in any possible way. Given two transition systems $T_1 = (S_1, L_1, s_0^1, \longrightarrow_1)$ and $T_2 = (S_2, L_2, s_0^2, \longrightarrow_2)$ we define the resulting interleaving system by $T_1 \parallel T_2 = (S_1 \times S_2, L_1 \cup L_2, (s_0^1, s_0^2), \longrightarrow)$, where $\longrightarrow$ is constructed using the structural operational semantics given in Table 2.1.

As evident, the state set of $T_1 \parallel T_2$ consists of all pairs (s_1, s_2) of states from the two subsystems and the transition relation allows any outgoing transition from either s_1 or s_2 from (s_1, s_2).

Another approach is based on communication between the systems and models for instance reactive systems where a system reacts to stimuli from another system or an environment. Here the operator is defined such that transitions cannot be taken independently, but must synchronise with a matching transition in another subsystem. Table 2.2 shows how to define $\longrightarrow$ in $T_1 \parallel T_2$ when taking a fully synchronising view. Here we are only allowed to take transitions when both subsystems can synchronise on the corresponding label.

Naturally one can also consider a mix between the two approaches described above, where some transitions can be taken independently and some must synchronise. Here one can define a set of handshaking actions H that must synchronise, while the remaining actions have the interleaving semantics. The transition relation is thus defined by the rules (A1) and (A2) for an action $a \notin H$ and by the rule (A3) for $a \in H$.

We will now investigate how a reasonable extension of transition systems can be defined in order to model quantities.

2.1.3. Modelling Resources

Before tempering with the model of transition systems presented above, let us consider all the relevant characteristics of the resources that we are about to model. This should help us make our model as useful as possible.

As noted in the motivation, we seek to model quantitative resources. These resources share some characteristics. Most of them can be both gained and consumed during an execution of a system. Temperature can for instance both increase and decrease, while a battery may or may not be rechargeable. Time is on the other hand a resource that cannot be regained. Furthermore, being able to compare two or more quantities of the same resource to each other is essential. For instance when formalising "the valve should open only if there is more than 4 gallons of oil left in the tank" or "pick the container with the least volume". By adding some sort of structure to the labels of transition systems, where such comparisons are possible, one can model resources.

Resources can be categorised as either continuous (e.g. time) or discrete (e.g. price), depending on the nature of the resource. Thus both a continuous and a discrete view can be necessary. An obvious choice when modelling discrete quantitative resources is to use an integer from $\mathbb{Z}$ to denote the quantity of the resource. The choice of $\mathbb{Z}$ as the domain seems to fulfil our initial idea of a set with a notion of an order (the $\leq$ relation) that can be both gained (positive integers) and consumed (negative integers). For a continuous resource, the set of reals, $\mathbb{R}$, is suitable instead. In the following we will use $\mathbb{Z}$ as the weight domain.

Where in the model should the quantities be placed? Since the resources are gained or consumed by the actions of the systems, it seems natural to impose a cost to system changes to denote how the resource is gained or consumed as the system performs its actions.

After the above discussion, we have the following natural extension to Definition 1.

Definition 3 (Weighted transition system)**.** A weighted transition system is a transition system $T = (S, L, s_0, \longrightarrow)$, where $\longrightarrow \subseteq S \times L \times \mathbb{Z} \times S$ is a transition relation with a label and an integer attached to each transition.

Along with the labels added in transition systems, we here decorate each transition also with an integer. We call the element added to each transition the *weight* of the transition. Again we write $s \xrightarrow{a,w} t$ if $(s, a, w, t) \in \longrightarrow$.

It is worth noticing that both $\mathbb{Z}$ and $\mathbb{R}$ equipped with addition and multiplication are special cases of the algebraic structure called a *semiring*. In the literature, weights are usually added by using semirings instead of integers; consult e.g. the handbooks [DKV09, Sak09, Eil74]. The concept of weighted automata with weights drawn from semirings was originally introduced by Schützenberger in [Sch61]. The use of semirings of course makes the theory more general, but also makes the theory less transparent for a potential user.

As for transition systems, we seek to define an execution of a weighted transition system $T = (S, L, s_0, \longrightarrow)$. For this purpose we define a *configuration* as a pair (s, v),

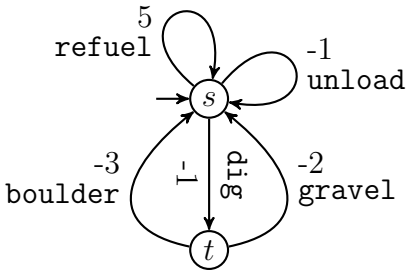


Figure 2.2.: An excavator modelled as a weighted transition system

$$\text{(A1)}\ \frac{s_1 \xrightarrow{a,\bar{w}}_1 t_1}{s_1 \parallel s_2 \xrightarrow{a,\bar{w}} t_1 \parallel s_2} \qquad \text{(A2)}\ \frac{s_2 \xrightarrow{a,\bar{w}}_2 t_2}{s_1 \parallel s_2 \xrightarrow{a,\bar{w}} s_1 \parallel t_2}$$

Table 2.3.: Interleaving semantics

where $s \in S$ and $v \in \mathbb{Z}$. An execution (or a weighted run) of T is then a finite or infinite sequence of configurations $(s_0, v_0), (s_1, v_1), \ldots$, such that for all $i \geq 0$ we have $s_i \xrightarrow{a_i, w_i} s_{i+1}$ and $v_{i+1} = v_i + w_i$ for some $a_i \in L$. By $\mathsf{runs}(s_0, v_0)$ we denote the set of all maximal weighted runs starting from (s_0, v_0).

Example 4. For the excavator from Example 2 fuel is surely a limited resource that the machine depends on, as it cannot run on an empty tank. We can model the fuel consumption (negative numbers) and gain (positive numbers) of a running excavator by modelling the excavator as a weighted transition system, as seen in Figure 2.2. The only action increasing the fuel level is `refuel`, the remaining ones all decrease the fuel level. Handling a `boulder` requires more fuel than `gravel`. A possible weighted run of the excavator is $(s, 0), (s, 5), (t, 4), (s, 1), (s, 0)$.

2.1.4. Composing Weighted Systems

As argued in Section 2.1.2, systems are often concurrent and it is therefore important that a model of the system can express this parallelism. Consider two weighted (and labelled) transition systems, $T_1 = (S_1, L_1, s_0^1, \longrightarrow_1)$ and $T_2 = (S_2, L_2, s_0^2, \longrightarrow_2)$. Again we may either adopt an interleaving or a synchronising view. In both cases we must decide how to handle the weights of the transitions of $T_1 \parallel T_2 = (S_1 \times S_2, L_1 \cup L_2, (s_0^1, s_0^2), \longrightarrow)$ when constructing the composite system. In case of an interleaving view, the most natural solution is to keep the weight of the original transitions, as each transition of the composed system corresponds to exactly one transition in one of the subsystems. The corresponding semantics of $\longrightarrow$ can be seen in Table 2.3.

A more interesting case occurs when assuming a fully synchronising view. Consider two systems each gaining and consuming the same resource. When run in parallel, what

$$\text{(A3)}\ \frac{s_1 \xrightarrow{a,\bar{w}_1}_1 t_1 \qquad s_2 \xrightarrow{a,\bar{w}_2}_2 t_2}{s_1 \parallel s_2 \xrightarrow{a,\bar{w}_1 \oplus \bar{w}_2} t_1 \parallel t_2}$$

Table 2.4.: Synchronising semantics

should happen to the use of the resource? The answer is surely dependent on the actual nature of the resource. As seen in Table 2.4 we define an operator $\oplus$, that given two weights of the subsystems determines the weight of the synchronised transition. How should $\oplus$ be defined? In case time is the resource and the weight of a transition denotes the time it takes to carry out the action, $w_1 \oplus w_2$ could be defined as $\max(w_1, w_2)$, since the total time of the synchronising action intuitively corresponds to the time of the slowest of the two actions. However, if the resource represented is the price, we rather define $w_1 \oplus w_2 = w_1 + w_2$, since this corresponds to the total price of the two transitions. How to define $\oplus$ is therefore entirely determined by the concrete resource. The definition of a structural composition operator for weighted systems is addressed in Paper D.

Having now defined the basic quantitative model used in the thesis, we move on to another important issue of model checking—how to express the requirements that we want to check.

2.2. Specification Phase

Model checking consists, as mentioned in Section 1.1, both of a modelling and a specification phase. The requirements of a system can be formalised in many ways, as described in Section 1.4. One way is a behavioural approach, where we use the same formalism for both specification and implementation and then check whether or not the behaviour of the two transition systems (in our case) are related by some suitable behavioural preorder. Many of these are organised in van Glabbeek's hierarchy [vG90]. However, as a transition system fully describes the possible behaviour of a system, we may find it easier to express our specification using a logical formalism. These formalisms express precisely the properties that the system must satisfy without restricting the behaviour of the system in other ways. Since the logical approach is the dominating approach to specifications in the world of model checking, we will take a closer look at some often applied logical formalisms for expressing specifications.

2.2.1. Logical Formalisms

Let us first try to outline two relevant types of properties one usually asks about a system and present some formalisms incorporating them. Afterwards we try to adapt some of these to a quantitative setting useful e.g. for embedded systems. The partition presented below is based on the view taken in [BK08] and might be defined differently in other literature.

Safety Safety properties can popularly be described as 'something bad should never happen'. An example is 'the system is deadlock-free' or for a mutual exclusion algorithm a relevant safety property would be 'only one process can enter the critical section at a time'.

More formally a safety property is satisfied if the system satisfies the property no matter what execution is conducted. Thus a safety property can be violated by providing a finite execution breaking the property at some point. Proving a safety property requires checking all possible executions.

Liveness Liveness properties are in some sense dual to safety properties. Notice that a safety property can be satisfied by a system with no behaviour at all. This is not the case for liveness properties, on the contrary, they require that progress is made in order to satisfy the property. An example of a liveness property for the mutual exclusion algorithm is that 'each process will eventually enter the critical section'.

A liveness property can only be violated in infinite time, since it for any finite execution of a system might be possible to extend the execution to a point where the property is true. Verification of a liveness property requires that all executions at some point satisfy the property.

In connection with liveness properties we sometimes make use of a *fairness* condition. In the example of the liveness property for the mutual exclusion algorithm, we see that the property may not hold in case one process enters the critical section infinitely often or just stays in the critical section, while the remaining processes wait. Such an execution is not 'fair' to the waiting processes and the problem can be circumvented by proposing a fairness condition. We can formulate a fairness condition saying that 'if some action is infinitely enabled, it should at some point be executed'. As a fairness condition can be used for ruling our unrealistic behaviour, it is sometimes assumed in order to prove a liveness property.

The safety and liveness properties described above are all temporal properties. However, by using the words 'eventually', 'always' or 'after this, then that' in the properties, we do not explicitly need to refer to some model of time. Such properties are often described using temporal logics [Pnu77, CE82, MP92], which are therefore the dominating logics when specifying properties for model checking purposes.

Temporal properties can furthermore be divided into two categories, *linear* or *branching* time properties. A linear-time view reasons on fixed runs, while a branching time view reasons on the whole tree of possible runs. This corresponds to the difference between two of the behavioural preorders; trace equivalence and bisimulation.

We now mention a few prominent logical formalisms for specifying the temporal properties described above. For linear-time properties, the logic called linear temporal logic (LTL) is often used [Pnu77]. A similar logic reasoning on branching-time properties is the logic computation tree logic (CTL) [CE82], that extends the Hennessy-Milner logic (HML) [HM85] with the until operator.

Other well-known temporal logics such as CTL* [EH86] or and the modal μ-calculus [Koz83] are also commonly used. The logic CTL* in fact includes both CTL

$$
\begin{array}{lll}
s \models \mathsf{true} & & \\
s \models a & \text{iff} & a \in p(s) \\
s \models \varphi_1 \wedge \varphi_2 & \text{iff} & s \models \varphi_1 \text{ and } s \models \varphi_2 \\
s \models \neg\varphi_1 & \text{iff} & s \not\models \varphi_1 \\
s \models \mathsf{EX}\varphi & \text{iff} & \exists s \longrightarrow t : t \models \varphi \\
s \models \mathsf{AX}\varphi & \text{iff} & \forall s \longrightarrow t : t \models \varphi \\
s \models \mathsf{A}(\varphi_1 \mathsf{U} \varphi_2) & \text{iff} & \forall s_0 \longrightarrow s_1 \longrightarrow s_2 \longrightarrow \ldots \in \mathsf{runs}(s) : \exists i \geq 0 : \\
 & & (s_i \models \varphi_2 \text{ and } \forall j \in \{0, \cdots, i-1\} \ : \ s_j \models \varphi_1) \\
s \models \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2) & \text{iff} & \exists s_0 \longrightarrow s_1 \longrightarrow s_2 \longrightarrow \ldots \in \mathsf{runs}(s) : \exists i \geq 0 : \\
 & & (s_i \models \varphi_2 \text{ and } \forall j \in \{0, \cdots, i-1\} \ : \ s_j \models \varphi_1)
\end{array}
$$

Figure 2.3.: Semantics of CTL

and LTL, while the modal μ-calculus includes all the aforementioned logics. The reason for not always choosing the most expressive logic when looking for a formalism to use, is due both to simplicity and hence understandability of the logic, but mainly to the fact that often a more expressive logic leads to a higher computational complexity.

As the contributions of Paper C and E treat the logic CTL, we will emphasise the syntax and semantics of CTL. The logical formulae of CTL are interpreted over states of a transition system, $T = (S, L, s_0, \longrightarrow)$, where each state is decorated with a subset of a set of atomic propositions. Let $p : S \to 2^A$ be the function assigning atomic propositions from the set A to each state. The syntax of the CTL formulae is given by the grammar

$$\varphi ::= \mathsf{true} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \mathsf{EX}\varphi \mid \mathsf{AX}\varphi \mid \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2) \mid \mathsf{A}(\varphi_1 \mathsf{U} \varphi_2) \ ,$$

where $a \in A$ is an atomic proposition. The semantics are given in Figure 2.3.

The well-known logical and temporal operators $\vee$, $\implies$, F (future) and G (globally) can easily be constructed using the given syntax:

$$
\begin{array}{ll}
\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2) & \varphi_1 \implies \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2 \\
\mathsf{EF}\varphi \equiv \mathsf{E}(\mathsf{true}\mathsf{U}\varphi) & \mathsf{AF}\varphi \equiv \mathsf{A}(\mathsf{true}\mathsf{U}\varphi) \\
\mathsf{EG}\varphi \equiv \neg\mathsf{AF}\neg\varphi & \mathsf{AG}\varphi \equiv \neg\mathsf{EF}\neg\varphi
\end{array}
$$

As the formalism of transition systems has been extended to incorporate quantitative aspects, it seems natural to also extend the specification formalisms as well, notably CTL.

2.2.2. Quantitative Requirements

Quantitative logics have been proposed previously in the literature [DG07, Mei09, BG09, FLT10], allowing us to state requirements about the resources of the system. We here

discuss the approach taken in Paper E, where we define a quantitative version of CTL.

Our approach is to remove the atomic propositions in the states and only reason on the accumulated weight along the runs. For most resources, it is natural to add the weights encountered along a run, as the accumulated weight at a point along the run represents the remaining quantity of the resource at that particular moment. For this reason we propose to replace atomic propositions with quantitative constraints on the form $v \bowtie c$, $\bowtie \in \{\leq, <, =, >, \geq\}$ and $c \in \mathbb{Z} \cup \{-\infty, \infty\}$, where v denotes the accumulated weight so far. With this version, reasoning only on the accumulated weights along runs, we can express a requirement like "along no run should the accumulated weight get below 0" as $\mathsf{AG}(v \geq 0)$.

Having a logic capable of expressing a wide range of properties is in general very useful. However, some properties regarding quantities are more applicable than others, and some are even so relevant that they should be treated separately. We will motivate such a property below.

For embedded systems, the access to resources can be sparse. Many everyday software systems rely on a energy source that can be exhausted, e.g. a battery. Many industrial machines control containers holding some material that should be used as a component in some production. The aim of the machine is to make sure that the container never runs out of material. This task is nowadays often carried out by a software controller. Other systems adjust the temperature or pressure of some environment and must take care that this value does not enter some 'bad' region. Common for these examples is that the resource in question must be kept above some threshold (often just nonnegative), and in some cases also below some upper threshold. This requirement must be met at all times during the execution of the system. The nature of the examples indicates that the current value of the resource can be modelled by simply adding the previous different fluctuations of the resource during the execution, given that we know the starting value. This will be useful when formalising this so far loosely described requirement.

According to the classification in Section 2.2.1, the property described above is a safety property, since violating the property can be done by presenting a finite run that violates the bounds.

As mentioned in the motivation, many systems do not terminate, but must keep on functioning for as long as needed. It is thus relevant that the requirement described in the previous paragraph is satisfied not only until some specific criteria is met (e.g. reaching some specific state), but also for infinite executions. In short we are looking for infinite runs subject to resource constraints.

The property can be phrased using the quantitative version of CTL mentioned earlier. For instance the requirement "the existence of a run that keeps the quantity of the resource between the lower bound ℓ and the upper bound u at all times" can be expressed with $\mathsf{EG}(v \geq \ell \wedge v \leq u)$. We can instead require that all runs should satisfy the weight constraints by changing the E to an A in the proposed formulae.

Driven by the examples described previously, a different weak upper bound can be formalised. If we consider the battery with a minimum and maximum capacity, we notice that when using the battery the maximum capacity cannot actually be exceeded, but will stay at the maximum capacity when trying to charge a fully charged battery.

This phenomenon is referred to as a *weak upper bound*. A weak upper bound can never be exceeded and weights above the bound will be truncated when constructing the weighted runs of the system. Thus we can impose two types of bounds—a (strict) bound and a weak bound. These bounds can additionally both be upper and lower bounds. However, a weak bound does not impose any non-trivial decision problems without used in connection with a strict bound. Furthermore any lower bound can be transformed into an upper bound and vice versa by multiplying all weights and bounds by -1. This implies that only three combinations of bounds are necessary to study: A lower bound, a lower and an upper bound, and a lower and a weak upper bound.

Due to its relevance, this thesis will specifically study the property of finding infinite runs subject to resource constraints in Paper A and B. However, we also propose different general logics for quantitative requirements in both Paper C, D and E. Apart from the weak upper bound, the logics of Paper C and E are also capable of expressing the specific property discussed above. These logics are furthermore interpreted on the specification formalism of modal transition systems, which we will discuss in Section 3.2.

2.3. Multiple Quantities

Another strikingly relevant feature of embedded systems (or other systems reasoning on resources) is the fact that one resource is very rarely enough. A system might both want to keep track of a battery, some container holding a material and the temperature. For this system, the model presented in Definition 3 cannot be used as a modelling formalism. A simple way to resolve this problem is to add not only singletons to each transition, but a vector. In this way each resource can be modelled using a coordinate of the vector and the dimension of the vector corresponds to the number of resources to be modelled.

Definition 5 (Multiweighted transition system)**.** A k-multiweighted (or just k-weighted) transition system is a transition system $T = (S, L, s_0, \longrightarrow)$, where $\longrightarrow \subseteq S \times L \times \mathbb{Z}^k \times S$ is a transition relation with an integer vector of dimension k attached to each transition.

Let $\bar{v}$ be a vector in $\mathbb{Z}^k$. By $\bar{v}[i]$ we denote the ith coordinate in $\bar{v}$. A configuration and a weighted run is defined similarly as for the case of $k = 1$, We define the addition of two vectors as the coordinate-wise sum, $(\bar{v_1} + \bar{v_2})[i] = \bar{v_1}[i] + \bar{v_2}[i]$ for $v_1, v_2 \in \mathbb{Z}^k$ and $1 \leq i \leq k$. Furthermore we write $\bar{v_1} \leq \bar{v_2}$ if $\bar{v_1}[i] \leq \bar{v_2}[i]$ for all $i \in \{1, \ldots, k\}$.

The quantitative version of CTL considered in Section 2.2.2 can easily by extended in order to handle multiple resources. One way to do this is by changing the syntax of the quantitative aspects of the logics such that one has to explicitly state the coordinate for which the constraint should hold.

What properties are useful to check for these multiweighted transition systems? The infinite runs subject to resource constraints property addressed in the Section 2.2.2 can surely we extended to a multiweighted setting, where we require that the accumulated

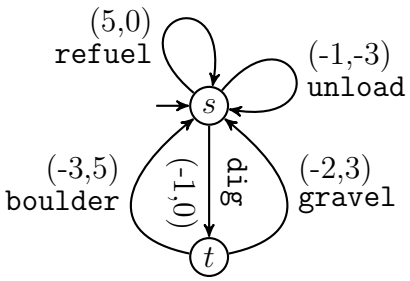


Figure 2.4.: An excavator modelled as a 2-weighted transition system

weight of all coordinates must remain nonnegative and below some fixed upper bound (given as a vector).

Example 6. The excavator from Example 2 was in Example 4 extended with quantitative information modelling the fuel consumption and fuel gain of each possible action. However, fuel may not be the only resource worth modelling. The weight of the material currently on the dipper is also important, as the excavator cannot carry a too heavy load. Modelling the excavator as a 2-weighted transition system as seen in Figure 2.4 gives the opportunity to model both resources. The pairs depicted in the figure represents the fuel consumption (first coordinate) and the change in weight currently loaded on the dipper (second coordinate). We notice that `refueling` and `digging` does not change the dipper load, while loading `gravel` is not as heavy as loading a `boulder` onto the dipper. The only way to reduce the weight on the dipper is to `unload`.

We can now state quantitative properties that we wish our excavator to satisfy. A reasonable property is that starting with no fuel and no load, there exists way of operating the excavator such that the amount of fuel in the tank is always between 0 and 5 and the load of the dipper is always between 0 and 10. Using the second version of the quantitative CTL adapted to a multiweighted setting, we can state this property as $\mathsf{EG}(\bar{v}[1] \geq 0 \wedge \bar{v}[1] \leq 5 \wedge \bar{v}[2] \geq 0 \wedge \bar{v}[2] \leq 10)$. The property is satisfied for the excavator, as the loop $(s,(0,0)),(s,(5,0)),(t,(4,0)),(s,(1,5)),(s,(0,2)),(s,(5,2)),(t,(4,2)),(s,(1,7)),\ (s,(0,4)),\ (s,(5,4)),\ (t,(4,4)),\ (s,(2,7)),\ (s,(1,4)),\ (s,(0,1)),\ (s,(5,1)),(t,(4,1)),(s,(1,6)),(s,(0,3)),(s,(5,3)),(t,(4,3)),(s,(2,6)),(s,(1,3)),(s,(0,0))$ adhere to the resource constraints.

For the case of $k = 1$ the infinite runs subject to resource constraints property has been studied in [BFL+08]. In Paper A we extend the work in [BFL+08] to a multiweighted setting. We here investigate the consequences in terms of decidability and complexity when adding additional weights to the infinite runs subject to resource constraints property.

When considering these infinite runs subject to resource constrained properties, another question arises. Assume that the upper bound is unknown. In this case it would be interesting to construct the set of upper bounds that allows such infinite runs to exist.

This set may be characterised by the set of smallest incomparable vectors contained in the set, as any suitable vector naturally implies that any larger vector is also suitable. Notice that a set of smallest vectors is needed as we do not have a total order on $\mathbb{Z}^k$.

Such a set of smallest incomparable vectors is known as the *Pareto frontier*, while each element in the set is called *Pareto optimal* or *Pareto efficient*. The notion of Pareto optimality originally stems from economics [Par71, Cir79] where it for instance describes an allocation of resources where no individual can change the allocation without making another individual worse off. Given this set, the suitable choice of upper bound is found by weighing the different resources and choosing the more optimal bound for the purpose. In this way the material used and money spend for constructing the actual system may be minimized, as the bounds can be minimised. We investigate the problem of finding the Pareto frontier in Paper B.

3 Adapting to the Challenges of Embedded Systems

In the previous chapter we have established a basic model for modelling resources and identified a class of properties that are relevant for resource dependent systems such as embedded systems.

As advocated by Henzinger and Sifakis in [HS06], designing embedded systems can be challenging, as hardware, software and environmental aspects cannot be separated and all of these concerns should therefore be incorporated into the design. In this chapter we will identify three aspects of embedded systems that are worth considering in relationship with quantities. We will investigate infinite runs subject to resource constraints and general quantitative logics in these new settings.

As a first aspect, we will address game theory, as games are suitable for modelling the reactive behaviour of embedded systems. Furthermore, stepwise refinement and compositional reasoning are also needed to ensure a better design process and we therefore consider modal transition systems as a specification formalism for quantitative embedded systems. Another important issue is robustness. We seek to ensure that small perturbations of the input values cause small perturbations in the output values. For this purpose the boolean-valued result of model checking is not enough, and we instead attempt to define a real-valued model checking result.

3.1. Games

The modelling formalism of labelled transition systems is not suitable for synthesis purposes, as described in Section 1.3, since the transitions are not distinguished as controllable and uncontrollable. It is therefore not possible to distinguish the environment from the controllable systems when modelling the system (consult e.g. Figure 1.2). For this

purpose we need to make use of some concepts from the world of game theory.

Games can be defined in many ways, depending on the nature of the real world concept that we seek to model. In our setting, the game will consist of two players, namely the system and the environment, whose decisions corresponds to the actions that they are able to take at a given time. Our games are turn-based, implying that players take turn in performing their actions. The currect state decides whose turn it is. As we look for quantitative formalisms, we are also interested in annotating each action with the corresponding use of one or several different resources, as in the case of transition systems. The definition below extends the definition of a multiweighted transition system to incorporate the game aspect.

Definition 7 (Game)**.** A k-weighted turn-based game (or just a game) is a tuple $G = (S_1, S_2, s_0, \longrightarrow)$, such that S_1 and S_2 are disjoint sets and $T_G = (S_1 \cup S_2, s_0, \longrightarrow)$ is a k-weighted transition system. We say that S_1 is the set of states belonging to the system (Player 1) and S_2 is the set of states belonging to the environment (Player 2).

A play of a game $G = (S_1, S_2, s_0, \longrightarrow)$ unfolds by placing a pebble in s_0 and then letting Player 1 and 2 move the pebble around in the system following the transitions. When the pebble is in a state from S_1, Player 1 picks an outgoing transition to move the pebble along, when the state is from S_2, Player 2 picks an outgoing transition.

Configurations and weighted runs in G is defined as configurations and weighted runs in the corresponding weighted transition system T_G. We denote all infinite and maximal weighted runs in G starting from (s_0, v_0) by $\mathsf{runs}(s_0, \bar{v}_0)$. In order to keep track of the decisions made by each player (i.e. the transitions picked), we define the notion of a strategy. A *strategy* for Player $i \in \{1, 2\}$ is a function σ from each finite prefix of a weighted run in $\mathsf{runs}(s_0, \bar{v}_0)$ of the form $(s_0, \bar{v}_0), (s_1, \bar{v}_1), \ldots, (s_n, \bar{v}_n)$, where $s_n \in S_i$ to a configuration $(s_{n+1}, \bar{v}_{n+1})$ such that $(s_0, \bar{v}_0), \ldots, (s_{n+1}, \bar{v}_{n+1})$ is a prefix of some run in $\mathsf{runs}(s_0, \bar{v}_0)$. A strategy for a player thus chooses the next transition to take when in a state belonging to this player. The choice is dependent on all the previous choices of the game. A run $(s_0, \bar{v}_0), (s_1, \bar{v}_1), \ldots$ *respects* a strategy σ if $\sigma(s_i, \bar{v}_i) = (s_{i+1}, \bar{v}_{i+1})$ for all $i \geq 0$.

The important property of finding infinite runs subject to resource constraints can also be formulated in a game setting and is then known in the literature as an *energy game* or a *generalised energy game* in the multiweighted case. Here we do not look for appropriate runs, but for a strategy that can ensure that no matter how the opponent behaves, the property is still satisfied. In this thesis the term energy game will also refer to generalised energy games.

Definition 8 (Energy game)**.** Given a k-weighted game $G = (S_1, S_2, s_0, \longrightarrow)$ an (generalised) energy game asks whether there exists a strategy σ for Player 1 such that all weighted runs $(s_0, \bar{v}_0), (s_1, \bar{v}_1), \ldots \in \mathsf{runs}(s_0, \bar{0})$ respecting σ satisfies $0 \leq \bar{v}_i$ for all $i \geq 0$.

In the affirmative case we say that Player 1 *wins* the energy game (or that the game is *winning*).

Without loss of generality we assume that the starting value of the resource is always 0 (by setting $v_0 = 0$). This is possible since any other starting value v can be simulated by adding a new start state s to the system and a transition $s \xrightarrow{v} s_0$.

As described in Section 2.2.2, we in some cases not only want to keep the accumulated weights nonnegative, but also below some upper threshold. This upper bound may be weak or not. We will therefore consider two extension of Definition 8. Given some vector of upper bounds $\bar{b}$, we define an energy game with upper bound as an energy game were all weighted runs respecting σ must furthermore keep their accumulated weight vectors below or equal to $\bar{b}$. An energy game with a weak upper bound $\bar{b}$ is an energy game where all coordinates of the accumulated weight vectors are truncated if going above their corresponding coordinate in $\bar{b}$.

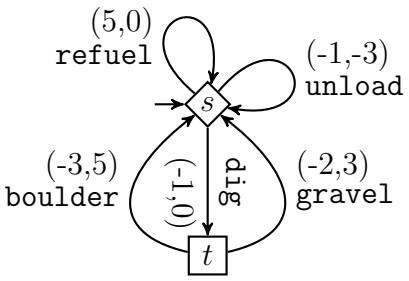
Energy games with only lower bounds were first considered by Chakrabarti et al. in [CdAHS03], while upper bounds were introduced in [BFL+08]. Other papers on energy games are [BFLM10, CD10, DDG+10]. Generalised energy games were studied in [BJK10, Cha10, CDHR10]. In Paper A we study generalised energy games with both lower and upper bounds, while Paper B studies the same games, but with an unknown upper bound.

Example 9. The running example of the excavator may also be modelled as a game. The model is seen in Figure 3.1a. Here the diamond state is controlled by Player 1, while the square state is controlled by Player 2. Since the driver of the excavator does not fully determine the content of the soil that the excavator digs in, it seems more realistic to model the excavator as a game, where Player 2 plays the role of the environment.

As a suitable energy game we seek a strategy for Player 1 such that the amount of fuel and volume of material on the dipper stay nonnegative and possibly below some upper bound. In case of no upper bounds, Player 1 can easily win by `refueling` indefinitely. Unfortunately, this behaviour is not useful in practice and will eventually flood the fuel tank. Let us therefore consider an energy game with the upper bounds used in Example 6 (the vector $(5, 10)$). As state t is uncontrollable to Player 1, the energy game cannot be won. Player 2 may choose `gravel` twice, leading to the path $(s, (0, 0)), (s, (5, 0)), (t, (4, 0)), (s, (2, 3)), (s, (1, 0), (t, (0, 0), (s, (-2, 3))$ violating the constraints.

However, using the upper bound $(8, 7)$ the energy game is winning. The winning strategy for Player 1 is seen in Figure 3.1b.

Two special cases of an energy game are found by setting either $S_1 = \emptyset$ or $S_2 = \emptyset$. In the former case we have no Player 1 states and therefore require that all runs must satisfy the boundary constraints. This is called a *universal* energy game. The latter case has no Player 2 states and thus look for the existence of a run satisfying the bounds. We call this an *existential* energy game.



(a) An excavator modelled as a 2-weighted game

if $\bar{w}[1] \geq 1 \wedge \bar{w}[2] \geq 3$ **then** `unload`

else if $\bar{w}[1] \leq 3$ **then** `refuel`

else `dig`

(b) Winning strategy for Player 1 with upper bound $(8, 7)$

Figure 3.1.: An energy game with upper bounds

Notice that universal and existential energy games can be defined on weighted transition systems, as the game aspect is not present in these special cases. Asking an AG property in CTL with the relevant bounds on the weights corresponds to a universal energy game, while an EG property corresponds to an existential energy game (as in Example 6).

3.2. Modal Transition Systems

Having established that the use of games is useful in order to capture the interactive behaviour of systems, we now turn to another relevant issue for model checking of embedded systems, namely the construction of a suitable specification formalism.

As mentioned in Section 1.4, the logical approach to specifications is suitable for stepwise refinement and logical composition of specifications, but is not the best solution when addressing the structural composition of specifications. Since structural composition is highly relevant for embedded systems, we choose the behavioural formalism of modal transition systems [LT88a, AHL+08] as specification formalism. This formalism is similar to normal transition systems, but allows for underspecification by giving rise to two types of transitions, namely the *may* and *must* transitions. The may transitions denote behaviour that may be left out in an implementation while the must transitions denote behaviour that must be present in an implementation. MTSs have a well-defined notion of satisfaction and refinement, and both structural and logical composition along with quotienting is possible [Rac08].

We will in the following enhance this theory with quantitative information, allowing for quantitative reasoning using modal transition systems. We begin by giving the standard definition of a modal transition system as found in [LT88a].

Definition 10 (Modal transition system)**.** A modal transition system (MTS) is a tuple $M = (S, L, s_0, \dashrightarrow, \longrightarrow)$, where S is a set of states, L is an alphabet, $s_0 \in S$ is the initial state and $\longrightarrow \subseteq \dashrightarrow \subseteq S \times L \times S$ are the must and may transition relations, respectively.

Given an implementation as a transition system and a specification as a modal transition system, one wants to check whether the implementation satisfies the requirements given by the specification. By providing the following refinement relation we are capable of stepwise refinement as advocated in [Lar89] where we gradually resolve the optional behaviour (the may transitions) present in the specification. A refinement of a specification is therefore a modal transition system that preserves the required behaviour of the original specification and does not introduce additional optional behaviour. By resolving each uncertainty we will end up with a system consisting of only required behaviour—an implementation. An implementation is therefore defined as a modal transition system where the may and must transitions coincide. Formally the refinement relation is as follows.

Definition 11 (Modal refinement for MTS)**.** Given two modal transition systems $M = (S, L_M, s_0, \dashrightarrow_M, \longrightarrow_M)$ and $N = (T, L_N, t_0, \dashrightarrow_N, \longrightarrow_N)$ we say that M modally refines N, written $M \leq_m N$ if there exists a relation $R \subseteq S \times T$ such that $(s_0, t_0) \in R$ and for all $(s, t) \in R$

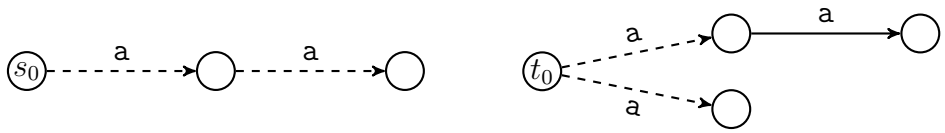
- if $t \xrightarrow{a}_N t'$ then also $s \xrightarrow{a}_M s'$ and $(s', t') \in R$, and
- if $s \overset{a}{\dashrightarrow}_M s'$ then also $t \overset{a}{\dashrightarrow}_N t'$ and $(s', t') \in R$.

For two implementations the notion of modal refinement corresponds to the behavioral equivalence of *bisimulation* [Par81, Mil83], which entails that any transition enabled in either of the systems must be matched by the other system yielding another bisimilar pair of systems.

Semantically we view an MTS M as the set of implementations refining M. We therefore say that M semantically (or *thoroughly*) refines another MTS N if the set of implementations of M is included in the set of implementations of N. The notion of modal refinement is sound with respect to this view. Thus an implementation being a modal refinement of a specification M is also an implementation of any other modal transition system N, where $M \leq_m N$. Unfortunately, modal refinement is not complete with respect to the semantic view as showed in [LT88a, HL89]. To illustrate this, Figure 3.2b and 3.2a (overtaken from [BKLS09b]) depict two specifications M and N, where M thoroughly refines N, but where $M \not\leq_m N$. However, completeness is ensured when considering deterministic systems [BKLS09b]. Determinism in modal transition systems is also treated by Henzinger and Sifakis in [HS06, HS07], where they furthermore advocate the use of determinism, as most practical systems are deterministic by nature.

Despite this shortcoming of modal refinement, this notion is often preferred to thorough refinement since the computational complexity of checking modal refinement is much lower than checking thorough refinement. Modal refinement requires only polynomial time (by a standard greatest fixed-point computation) while thorough refinement is EXPTIME-complete [BKLS09a].

That modal transition systems constitute a flexiable graphical specification formalism [BL92], is evident from the wide range of applications in different context such



(a) N (b) M

Figure 3.2.: Modal refinement is not complete

as product lines [FUB06, LNW07b, GLS08b], interface theories [UC04, RBB+09] and modal abstractions in program analysis [GHJ01, HJS01, NNN08].

There are several extentions and variants of the formalism. This includes timed modal specifications [CGL93], the probabilistic extension of abstract probabilistic automata [DKL+11], and specifications with explicit presentation of data [BLL+11]. Expanding on the expressivitity of the may and must transisition, the variants of disjunctive MTS [LX90], 1-selective MTS [FS08], MTS with obligations [BK11], and parametric MTS [BKL+11] have been introduced. In the following section we extend the original MTSs to incorporate the use of resources, resulting in a specification formalism for multiweighted transition systems as defined in Definition 5.

3.2.1. Refinable Sets of Labels

The approach used when extending normal transitions systems to a quantitative setting was to add an integer weight (possibly a vector of integer weights) to each transition. This rather simple approach is in Paper C extended to the modal setting. As the may and must transitions allow for a loose specification giving rise to several different concrete implementations, it is natural to also take this approach when adding quantitative information to MTSs. Thus the quantities should have a flavor of "looseness" as well. This can be achieved by adding intervals of integers to each transition instead of integers. In this way a designer does not have to specify the precise weight of the transition when constructing the specification, but can specify an interval wherein the exact quantity must lie, leaving the exact choice of the weights to be determined when constructing the implementation. Let $\mathcal{I}$ be the set of all non-empty intervals $[n, m] = \{a \in \mathbb{Z} \mid n \leq a \leq m\}$ such that $n \leq m$ and $n, m \in \mathbb{Z} \cup \{\infty, -\infty\}$.

Definition 12 (Weighted modal transition system)**.** A weighted modal transition system (WMTS) is a MTS, $M = (S, L, s_0, \dashrightarrow, \longrightarrow)$, where $\longrightarrow \subseteq \dashrightarrow \subseteq S \times L \times \mathcal{I} \times S$ assigns an action and an interval to each transition.

An implementation is obtained by picking one concrete weight in each interval and resolving the may transitions. An implementation thus corresponds to a weighted transition system as defined in Definition 3.

The refinement relation must be updated in order to deal with this new information on the transitions. As we would like a refinement to be closer to a real implementable

system with no uncertainties, it is natural to require that for matching transitions the refinement carries a possibly more narrow interval still contained in the interval of the refined transition. In this way we narrow the set of available integers for each weight. An implementation will settle on a concrete weight for each transition. Formally we extend Definition 11 as follows.

Definition 13 (Modal refinement for WMTS)**.** Given two weighted modal transition systems $M = (S, L_M, s_0, \dashrightarrow_M, \longrightarrow_M)$ and $N = (T, L_N, t_0, \dashrightarrow_N, \longrightarrow_N)$ we say that M modally refines N, written $M \leq_m N$ if there exists a relation $R \subseteq S \times T$ such that $(s_0, t_0) \in R$ and for all $(s, t) \in R$

- if $t \xrightarrow{a,J}_N t'$ then also $s \xrightarrow{a,I}_M s'$ such that $I \subseteq J$ and $(s', t') \in R$, and
- if $s \overset{a,I}{\dashrightarrow}_M s'$ then also $t \overset{a,J}{\dashrightarrow}_N t'$ such that $I \subseteq J$ and $(s', t') \in R$.

As in the case of multiweighted transition systems, the theory of weighted modal transition systems can be extended to a k-weighted setting by considering vectors of intervals drawn from $\mathcal{I}^k$ instead of $\mathcal{I}$. This approach is taken in Paper E.

Example 14. Imagine that the 2-weighted implementation M of an excavator seen in Figure 2.4 was build from the specification N of a family of excavators given in Figure 3.3. Here the solid transitions indicate a must transition (and an underlying may transition), while the dashed lines indicate a may transition. The must transitions of N can be matched by the corresponding must transitions in M, while the may transitions of M also have matching may transitions in N. Furthermore each weight of the implementation is a member of the corresponding interval from the specification. Therefore $M \leq_m N$ (M is even an implementation of N). Different implementations could have been obtained by either picking different weights from any of the intervals or by implementing none or only one of the may transitions, as picking up any material after `digging` is not required.

The choice of intervals as the used notion for specification of quantitative information is due to the low complexity when handling intersection and set inclusion as necessary when studying refinement and logical composition. However, a more general approach would be applicable for systems where the quantitative information cannot be expressed as intervals of integers, but rather as a set of integers or a general lattice.

For a good specification formalism we are interested in expressing "loose" information that may be refined into concrete implementable information. For this purpose partially ordered sets are suitable, as they impose an order on the elements of a set suitable for the refinement relation, but do not restrict the elements of the set further. We therefore define a *label-set* as a set of labels with a partial order $(K, \sqsubseteq)$, where $\bot \in K$ is the least element of K. The element $\bot$ is included to model inconsistencies when reasoning on the conjunction and quotient of two systems. The intuition is that we use

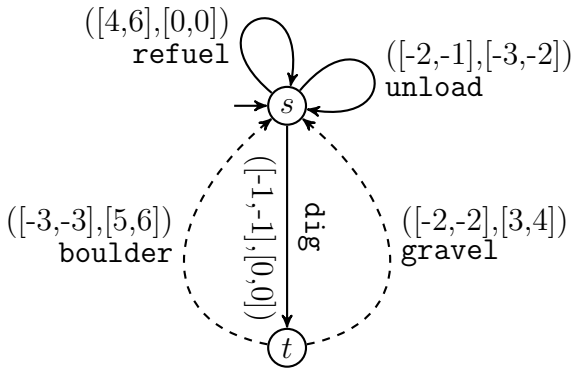


Figure 3.3.: An excavator modelled as a weighted modal transition system N

the smallest elements (apart from $\bot$) to denote implementable labels, while all other elements above them correspond to unresolved uncertainties regarding the label. The set of implementation labels in K is denoted $\mathrm{Imp}(K)$ and consists of all $k \in K \setminus \bot$ such that $k' \sqsubseteq k$ implies $k' = k$ for all $k' \in K \setminus \bot$. To each label $k \in K$ we associate the set $[\![k]\!] = \{k' \in \mathrm{Imp}(K) \mid k' \sqsubseteq k\}$, denoting the set of implementation labels below k.

We now the following defintion.

Definition 15 (Label-structured modal transition system)**.** A label-structured modal transition system (LSMTS) is a tuple $M = (S, (K, \sqsubseteq), s_0, \dashrightarrow, \longrightarrow)$, where S is a set of states, $(K, \sqsubseteq)$ is a label-set, $s_0 \in S$ is the initial state and $\longrightarrow \subseteq \dashrightarrow \subseteq S \times K \times S$ assigns a label to each transition.

As evident, we draw the weights from K instead of $\mathbb{Z}$ or $\mathbb{Z}^k$ as in the weighted case. In order to construct only sensible systems we require that the label-sets are *well-formed*, meaning that $[\![k]\!] \neq \emptyset$ for all $k \in K \setminus \{\bot\}$, such that any specification always guarantees some implementation.

For the refinement relation for LSMTSs, we require that a transition in the refinement has a label smaller than the label of the corresponding refined transition. The two items in Definition 11 are therefore updated to accommodate this requirement.

Definition 16 (Modal refinement for LSMTS)**.** Given two LSMTSs $M = (S, (K_M, \sqsubseteq_M), s_0, \dashrightarrow_M, \longrightarrow_M)$ and $N = (T, (K_N, \sqsubseteq_N), t_0, \dashrightarrow_N, \longrightarrow_N)$ we say that M modally refines N, written $M \leq_m N$ if there exists a relation $R \subseteq S \times T$ such that $(s_0, t_0) \in R$ and for all $(s, t) \in R$

- if $t \xrightarrow{k}_N t'$ then also $s \xrightarrow{\ell}_M s'$ such that $\ell \sqsubseteq k$ and $(s', t') \in R$, and
- if $s \overset{\ell}{\dashrightarrow}_M s'$ then also $t \overset{k}{\dashrightarrow}_N t'$ such that $\ell \sqsubseteq k$ and $(s', t') \in R$.

An implementation is an LSMTS where $\longrightarrow = \dashrightarrow$ and for any $s \xrightarrow{a} s'$ we have $a \in \mathrm{Imp}(K)$. With this more general set of labels, it is possible to capture the interval

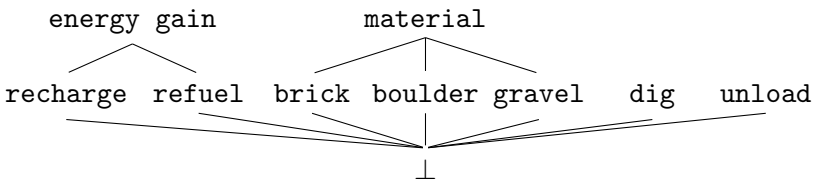


Figure 3.4.: $\sqsubseteq$

weights presented previously, but also any general set of integers (with $\subseteq$ as the partial order). Moreover K can represent a normal alphabet used for a modal transition system by letting each $k \in K \setminus \{\bot\}$ be an implementation label. Then no label refinement can take place and the labels thus act as a normal alphabet.

Example 17. The label-set depicted in Figure 3.4 (along with its ordering) can be useful when specifying a more general excavator. Here we allow not only the weights to be refined, but also the actions. A specification can have a transition labelled `energy gain`, indicating that either a `refuel` or `recharge` action should be present, but allowing the concrete implementation to decide its source of energy (fuel or battery). The material available for the excavator can also be unspecified in the specification by using the more general label `material`.

Paper D introduces the idea of refinable labels and defines label-sets. The definition of the satisfaction and modal refinement relation is presented and we prove that refinement is sound with respect to thorough refinement, but only complete in case of deterministic systems. Furthermore we define operators for structural and logical composition and for the quotient. The structural operator satisfy the property of compositional refinement, implying that we can refine systems before composing them. As expected, our conjunction operator yields the largest refinement refining all systems being logically composed (under suitable conditions). Our quotient operator is also proven to be sound and to construct the largest possible quotient, as one would expect.

The formalism of label-structured modal transition systems is therefore applicable as a specification formalism for embedded systems with quantitative resources. The formalism also allows labels to be refined, generalising the concept of refinement.

3.2.2. Logical Characterisation

We will now turn to the logical characterisation of multiweighted modal transition systems. Logics are excellent at expressing only the required behaviour without restricting the remaining behaviour in any way. This is also possible with an WMTS, but is however quite cumbersome, as one must explicitly add may transitions with any behaviour. Properties that e.g. no deadlock must occur or that some state is always reachable are easy to express using temporal logics, but may be more complex to express using WMTSs. We can therefore profitably use logical formulae alongside WMTSs when de-

signing a system. In [Lar89] the first logical characterisation of refinement was obtained using HML.

In this thesis we seek to define a quantitative version of CTL that reasons on WMTSs (in particular the multiweighted extension) instead of weighted transitions systems. Ideally we would like a specification to satisfy a logical formula if and only if all its implementations satisfy the specification. In this way we can propose a general specification for a system and check that it satisfies some requirements given in CTL. Hereafter we can safely construct any implementation, as this will also satisfy the logical formula.

How do we define such a logic in order to make it both *sound* and *complete* with respect to implementations? Here soundness implies that if a specification satisfies a formula then so will all refinements, while completeness implies that if all implementations satisfy a formula, then so does the specification. Surely we need a sound logic, as we otherwise might construct a flawed implementation even though the specification satisfies the logical formula. Completeness is useful as it allows for counterexample generation: a specification not satisfying a formula has at least one implementation also not satisfying it. For related work on generalised model checking and 3-valued logics, consult [BG00, HJS01, GHJ01, GP09].

In the folowing we will use the quantitative version of CTL from Section 2.2.2 (used in Paper E) for definitions and examples. The reasoning behind the semantic choices can however be applied to any version of CTL. As we here consider k-weighted modal transition systems, we extend the syntax of the accumulated weight logic to use linear expressions on the form $e ::= \langle i \rangle \cdot c \mid e + e$, where $0 \leq i \leq k$ and $c \in \mathbb{Z}$. With $\langle i \rangle$ we address the ith coordinate in the accumulated weight vector. Thus instead of atomic propositions we use propositions on the form $e \bowtie b$, where $\bowtie \in \{<, \leq, =, \geq, >\}$ and $b \in \mathbb{Z} \cup \{-\infty, \infty\}$.

We use a positive normal form of CTL that uses the temporal operators next (X), until (U) and weak until (W). The syntax of our accumulated-weight logic is given by the following grammar, where we distinguish between state and path formulae.

$$\varphi ::= e \bowtie b \mid \neg(e \bowtie b) \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \mathsf{E}\psi \mid \mathsf{A}\psi \ . \tag{3.1}$$

The path formulae are given by

$$\psi ::= \mathsf{X}\varphi \mid \varphi_1 \mathsf{U} \varphi_2 \mid \varphi_1 \mathsf{W} \varphi_2 \ .$$

The crucial point is how to define the semantics of the logic in order to obtain soundness and completeness. As we reason on WMTSs, we must distinguish may and must transitions when giving the semantics, since they are treated differently in the modal refinement relation.

Intuitively we should not consider the may transitions when dealing with a formula requiring the existence of a run, as any may transition can be dropped in an implementation. Thus a witnessing run containing may transitions is not guaranteed to appear in all implementations. On the other hand, for any formula where all runs must satisfy some property, we must study any run that might be a maximal run in some implementation.

$$
\begin{array}{lll}
(s,\bar{v}) \models e \bowtie b & \text{iff} & [\![e]\!]_{\bar{v}} \bowtie b \\
(s,\bar{v}) \models \neg(e \bowtie b) & \text{iff} & [\![e]\!]_{\bar{v}} \not\bowtie b \\
(s,\bar{v}) \models \varphi_1 \wedge \varphi_2 & \text{iff} & (s,\bar{v}) \models \varphi_1 \text{ and } (s,\bar{v}) \models \varphi_2 \\
(s,\bar{v}) \models \varphi_1 \vee \varphi_2 & \text{iff} & (s,\bar{v}) \models \varphi_1 \text{ or } (s,\bar{v}) \models \varphi_2 \\
(s,\bar{v}) \models \mathsf{E}\psi & \text{iff} & \exists\ \gamma = (s_0,\bar{v}_0),(s_1,\bar{v}_1),\ldots \in \mathsf{runs}(s,\bar{v}) \text{ where} \\
 & & s_k \longrightarrow s_{k+1} \text{ for all } k \geq 0 \ : \ \gamma \models \psi \\
(s,\bar{v}) \models \mathsf{A}\psi & \text{iff} & \forall\ \gamma = (s_0,\bar{v}_0),(s_1,\bar{v}_1),\ldots \in \mathsf{pruns}(s_0,\bar{v}_0) \ : \ \gamma \models \psi
\end{array}
$$

Figure 3.5.: Semantics of the accumulated-weight logic for WMTSs

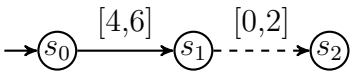


Figure 3.6.: Counterexample to completeness

This reasoning argues for the first try of defining the semantics, which is also proposed in Paper C.

In order to formally state the semantics of the logic, we define a *run* in a WMTS M as a sequence of configurations $(s_0,\bar{v}_0),(s_1,\bar{v}_1),\ldots$ such that for all $i \geq 0$, $s_i \overset{\bar{W}_i}{\dashrightarrow} s_{i+1}$ and $\bar{v}_{i+1} = \bar{v}_i + \bar{w}_i$, where $\bar{w}_i \in \bar{W}_i$. Again the set $\mathsf{runs}(s,\bar{v})$ denotes all maximal runs in M starting from $(s,\bar{v})$. By $\mathsf{pruns}(s,\bar{v})$ we denote the set of all runs from $(s,\bar{v})$ that are either infinite or has no outgoing must transitions. These paths are the ones that may be maximal runs in an implementation.

State formulae φ, φ_1 and φ_2 are interpreted over configurations of WMTSs as seen in Figure 3.5. The semantics for path formulae is unchanged from the normal interpretation over paths in MTSs and are therefore omitted.

The semantics specifies that in order for a formula quantified by E to be true, we only look for maximal runs of only must transitions. For formulae quantified by A we on the contrary need to check every outgoing path ending in a state with no outgoing must transitions in order to establish satisfaction.

For such a logic in positive normal form the distinction between may and must transitions for E and A formulae ensures soundness w.r.t refinement, as proven in Paper C.

Unfortunately, the logic presented in Paper C (and above) is not complete. A simple counterexample can be seen by considering the formula $\varphi = \mathsf{EG}(\langle 1\rangle \leq 10)$ and the 1-weighted WMTS seen in Figure 3.6. Since no maximal path with only must transitions exists, we can conclude that the configuration $(s_0,0)$ does not satisfy φ. On the other hand, all implementations have maximal paths consisting of either one or two must transitions, all meeting the requirement that the accumulated weight does not exceed 10 at any point.

This example shows that deciding the satisfiability of a formula quantified by E by only studying the maximal paths of must transitions is not adequate to ensure com-

pleteness. Even though we cannot be sure that one specific may transition will appear in all implementations, we can try to make sure that regardless of what may transitions are implemented the property is still satisfied. As this approach seems similar to the game theoretic approach where Player 1 seeks to ensure some behaviour regardless of the choices of Player 2, we will try to construct the semantics for the logic based on games in order to ensure completeness.

3.2.3. Completeness Using Games

In order to ensure that a WMTS satisfies a logical formula if and only if all implementations satisfy it, we can view WMTSs as games when dealing with a formula on the form $\mathsf{E}\psi$. As we do not know what may transitions will be implemented or what precise weights will be chosen in an arbitrary implementation, we let Player 2 control these uncontrollable choices while Player 1 controls the must transitions. We can then seek a strategy for Player 1 such that no matter what choices Player 2 makes (i.e. no matter what implementation we consider) we can still satisfy the formula. In this way we still guarantee soundness, since our strategy ensures satisfaction of the formula in any implementation chosen by Player 2.

We first notice that in the case of a state s with both outgoing may and must transitions, we do not have to consider the outgoing may transitions when checking that a formula on the form $\mathsf{E}\varphi$ is satisfied. This is evident if we consider the implementation I that does not implement the may transitions. Here if $\mathsf{E}\varphi$ is satisfied, some run γ exists that satisfies ψ. The run γ does for sure not stop in s, since it must be maximal. Therefore γ will also be a maximal and witnessing run for any implementation that also implements some or all of the outgoing may transitions from s. Thus we only distinguish states with outgoing must transitions (controllable by Player 1) and states with no outgoing must transitions (controllable by Player 2). As we also need the formula to hold for any choice of weights from the intervals, we leave this choice to Player 2. The game played here is therefore modified slightly from Definition 7, since we after each choice of a transition (by either Player 1 or 2) let Player 2 choose a weight from the weight interval. Furthermore Player 2 may stop the game in any state controllable by Player 2 in order to model that no outgoing may transitions are implemented.

We define a strategy for Player 1 in a WMTS as a function from each finite prefix of a weighted run that ends in a state s controllable by Player 1 to a must transition outgoing from s. A run $(s_0, \bar{v}_0), (s_1, \bar{v}_1), \ldots$ respects a strategy σ if for all s_i controllable by Player 1 we have $\sigma(s_i, \bar{v}_i) = (s_i, \bar{W}, s_{i+1})$ for some $\bar{W}$. We can now state the game based semantics for $\mathsf{E}\psi$.

$$(s, \bar{v}) \models \mathsf{E}\psi \quad \text{iff} \quad \begin{array}{l} \text{there exists a strategy } \sigma \text{ for Player 1 such that} \\ \text{for all } \gamma \in \mathsf{runs}(s, \bar{v}) \text{ where } \gamma \text{ respects } \sigma \text{ we have } \gamma \models \psi \ . \end{array}$$

The example depicted in Figure 3.6 along with the formula $\varphi = \mathsf{EG}(\langle 1 \rangle \leq 10)$ is no longer a counterexample to completeness when assuming the semantics based on games. This is evident since the (only available) strategy that maps $(s_0, 0)$ to $s_0 \xrightarrow{[4,6]} s_1$ is

sufficient to ensure that no matter what Player 2 does, the resulting run will always have an accumulated weight no larger than 10.

As evident, the game-based semantics rules out the counterexample to completeness presented above. However, this is unfortunately not always the case. Consider the formula $\varphi = \mathsf{AF}(\mathsf{AX}\ \mathsf{false})$. The formula states that for all paths, at some point there is no outgoing transitions. Notice that false can be modelled by $\langle 1 \rangle < 0 \wedge \langle 1 \rangle > 0$. Now for the example in Figure 3.6 we have $(s_0, 0) \not\models \varphi$, as $(s_0, 0), (s_1, 4)$ is a path in $\mathsf{pruns}(s_0, 0)$ where s_1 has an outgoing transition. However, any implementation will surely in all weighted runs encounter a state with no outgoing transition. In general the logic is therefore not complete.

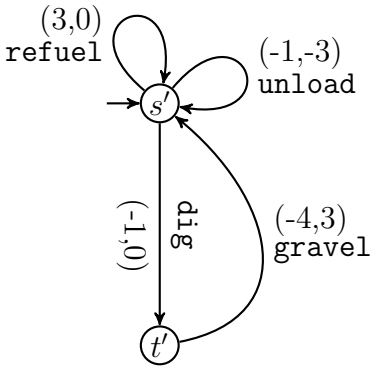
The semantics based on games is presented in Paper E for a small, but practically useful fragment of CTL. We here prove both soundness and completeness of the logic if we restric ourselves to the unnested formulae of the form $\mathsf{AG}\varphi$, $\mathsf{AF}\varphi$, $\mathsf{EG}\varphi$, and $\mathsf{EF}\varphi$.

3.3. Metrics

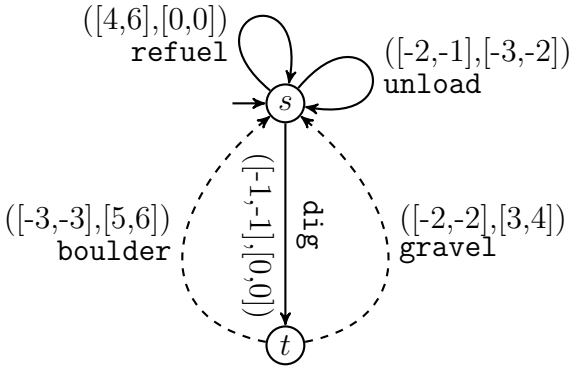
The model checking view presented in Figure 1.1 shows that the result of the model checking is either a 'yes' or a 'no'. This boolean answer is also used when model checking quantitative models to quantitative specifications. However, the quantities present in the specification and models pave the road for a quantitative answer to the model checking question instead of a boolean one. This will help engineers in their search for correct implementations, as they in the negative case are interested in some way of telling 'how close' they are to satisfying the specification. A *quantitative analysis* entails that two implementations not satisfying the specification can be compared and we might find that one is closer to satisfying the specification than the other. In the case that an implementation of the specification is not at hand, we are thus able to rate the implementations not satisfying the specification and pick the better match. This is not possible in the previously considered boolean framework, as two such implementations cannot be distinguished.

In order to develop a quantitative framework for analysis, we first need to define a notion of distance between two implementations. This is done by lifting the decision problem associated with the known behavioral equivalences and preorders into a search problem, assigning a nonnegative real number to pairs of systems. Quantitative analysis was recently studied for reactive systems by de Alfaro et al. [dAM01, dAF03, dAFS04, dAFS09] and Thrane et al. [TFL10, FLT10, FTL11, FLT11] and for probabilistic systems in [JS90, DGJP04]. An overview can be found in [Thr11]. Several distances have been considered—these merely model different ways of estimating the distance and are applicable in different settings.

Having defined a metric on implementations, it can be lifted to a distance from an implementation to a specification. This has been done for the logic of CTL [FLT10], timed CTL [HMP05] and the μ-calculus [LLM05] among others. Recently we have developed a framework for calculating the distance between a quantitative implementation



(a) Implementation M of an excavator



(b) Specification N of an excavator

$$d_m(s', s) = \max \begin{cases} 1 + \lambda\, d_m(s', s) \\ 0 + \lambda\, d_m(t', t) \end{cases}$$

$$d_m(t', t) = 2 + \lambda\, d_m(s', s)$$

(c) Calculating the modal refinement distance $d_m(M, N)$

Figure 3.7.: Modal refinement distance

and a quantitative modal transition system [BFJ+11][1]. In the paper we study the notion of modal and thorough refinement and broaden these notions to distances, giving a nonnegative real as the result. In this way we are both capable of measuring the distance between an implementation and a specification, but in general also between two specifications. The asymmetric thorough distance is defined on sets of implementations and for two modal transition systems, M and N, the thorough distance from M to N is smaller than ϵ if and only if for each implementation of M there exists an implementation of N such that the distance between these two implementations is smaller than ϵ. A distance of 0 thus implies that M thoroughly refines N. As hoped for, we show that the modal distance is never smaller than the thorough distance, implying that the modal distance can be used as an upper bound for the thorough distance. Thus the boolean framework, where modal refinement implies thorough refinement, generalises nicely to the quantitative framework.

Example 18. Figure 3.7a depicts an implementation M of an excavator. If we compare the implementation with the specification N given in Figure 3.7b, we see that the energy gain/consumption associated with the `refuel` and `gravel` actions in M does not fit into the intervals specified in N.

Using the modal refinement distance d_m defined in [BFJ+11] and a discounting factor of $\lambda = 0.9$, we find that the distance from M to N is $d_m(M, N) = 10$. This

[1] As this paper does not directly fit the subject of this thesis, it is not included as a paper.

follows since the distance from s' to s is calculated as the distance between either the two `refuel` or `dig` actions added to the discounted weight between the two new resulting states. Similarly for the distance from t' to t. The distance between the two `refuel` actions is 1 (the integer 3 has distance 1 to the interval $[4, 6]$), the distance between the two `dig` actions is 0 and the distance between the two `gravel` actions is 2. These intermediate results are seen in Figure 3.7c.

Finding the least fixed points of the equations yields $d_m(M, N) = d_m(s', s) = \max(1 + 0.9 \cdot 10, 0.9\,(2 + 0.9 \cdot 4.47)) = 10$. This gives an upper bound on the thorough refinement distance between the two systems—in fact the two distances coincide for this example since the systems are deterministic.

4 Thesis Summary

This chapter presents a summary of the five papers that are included in this thesis. For each paper the abstract, contributions and publication history are presented.

The layout of the papers has been edited to fit the format of this thesis and appendices has been moved to the main text.

The sub-questions identified as part of the research objectives in Section 1.5 are addressed in the papers as follows. Question i) is primarily treated in Paper C and D, suggestions for question ii) are given in Paper C, D and E, question iii) is studied in Paper A and B, question iv) is treated in Paper E, question v) is mainly addressed in Paper A and E, and finally question vi) is the focus of Paper B.

Paper A: Energy Games in Multiweighted Automata

Uli Fahrenberg Line Juhl Kim G. Larsen Jiří Srba

Energy games have recently attracted a lot of attention. These are games played on finite weighted automata and concern the existence of infinite runs subject to boundary constraints on the accumulated weight, allowing e.g. only for behaviours where a resource is always available (nonnegative accumulated weight), yet does not exceed a given maximum capacity. We extend energy games to a multiweighted and parameterized setting, allowing us to model systems with multiple quantitative aspects. We present reductions between Petri nets and multiweighted automata and among different types of multiweighted automata and identify new complexity and (un)decidability results for both one- and two-player games. We also investigate the tractability of an extension of multiweighted energy games in the setting of timed automata.

Contributions

- Definition of multiweighted energy games seeking infinite runs in multiweighted automata constrained by both upper and lower bounds on the accumulated weight.
- Complexity results for all identified subclasses of the problem relying on reductions to and from Petri nets and among different classes of energy games.
- (Un)decidability results for existential energy games where the upper bound is unknown.
- Undecidability result for timed multiweighted energy games.

Publication History

The paper was accepted and presented at the 8th International Colloquium on Theoretical Aspects of Computing (ICTAC'11) and published in Proc. of ICTAC'11, volume 6916 of LNCS, pages 95-115, Springer, 2011.

Paper B: Optimal Bounds for Multiweighted and Parametrised Energy Games

Line Juhl Kim G. Larsen Jean-François Raskin

Multiweighted energy games are two-player multiweighted games that concern the existence of infinite runs subject to a vector of lower and upper bounds on the accumulated weights along the run. We assume an unknown upper bound and calculate the set of vectors of upper bounds that allow an infinite run to exist. For both a strict and a weak upper bound we show how to construct this set by employing results from previous works, including an algorithm given by Valk and Jantzen for finding the set of minimal elements of an upward closed set. Additionally, we consider energy games where the weight of some transitions is unknown, and show how to find the set of suitable weights using the same algorithm.

Contributions

- Method for constructing the set of upper bounds that win an energy game with weak unknown upper bound employing a generic algorithm for calculating sets of minimal elements.
- Upper bound on the complexity for constructing the set of upper bounds that win an energy game with strict unknown upper bound.
- Method for constructing the set of weights of transitions that win an energy game with unknown weights of some transitions.

Publication History

Manuscript in preparation for submission.

Paper C: Modal Transition Systems with Weight Intervals

Line Juhl Kim G. Larsen Jiří Srba

We propose *weighted modal transition systems*, an extension to the well-studied specification formalism of modal transition systems that allows to express both required and optional behaviours of their intended implementations. In our extension we decorate each transition with a weight interval that indicates the range of concrete weight values available to the potential implementations. In this way resource constraints can be modelled using the modal approach. We focus on two problems. First, we study the question of existence/finding the largest common refinement for a number of finite deterministic specifications and we show PSPACE-completeness of this problem. By constructing the most general common refinement, we allow for a stepwise and iterative construction of a common implementation. Second, we study a logical characterisation of the formalism and show that a formula in a natural weight extension of the logic CTL is satisfied by a given modal specification if and only if it is satisfied by all its refinements. The weight extension is general enough to express different sorts of properties that we want our weights to satisfy.

Contributions

- Definition of weighted modal transition systems and their appurtenant refinement relation.
- PSPACE-completeness of deciding existence/finding the largest common refinement given a number of deterministic specifications.
- Syntax and semantics and proof of soundness for a weighted extension of CTL reasoning on weighted modal transition systems.

Publication History

An abstract was accepted and presented at the 21st Nordic Workshop on Programming Theory (NWPT'09). The full paper is published in Journal of Logic and Programming, volume 81(4), pages 408-421, Elsevier, 2012.

Paper D: Extending Modal Transition Systems with Structured Labels

Sebastian Bauer Line Juhl Kim G. Larsen
Axel Legay Jiří Srba

We introduce a novel formalism of label-structured modal transition systems that combines the classical may/must modalities on transitions with structured labels that represent quantitative aspects of the model. On the one hand, the specification formalism is general enough to include models like weighted modal transition systems and allows the system developers to employ more complex label refinement than in the previously studied theories. On the other hand, the formalism maintains the desirable properties required by any specification theory supporting compositional reasoning. In particular, we study modal and thorough refinement, determinization, parallel composition, conjunction, quotient, and logical characterization of label-structured modal transition systems.

Contributions

- Definition of label-structured modal transition systems allowing refinement of labels.
- Proof of desirable properties for the defined notions of determinization, modal and thorough refinement.
- Definition of the operators for structural composition, logical composition and quotienting and proof of desirable properties.
- Syntax and semantics of a label-structured extension of HML reasoning on label-structured modal transition systems.
- Proof of soundness in general and completeness for a suitable subset of the extended HML logic.

Publication History

The paper is published in Mathematical Structures in Computer Science, volume 22, pages 581-617, Cambridge University Press, 2012.

Paper E: A Logic for Accumulated-Weight Reasoning on Multiweighted Modal Automata

Sebastian Bauer Line Juhl Kim G. Larsen
Axel Legay Jiří Srba

Multiweighted modal automata provide a specification theory for multiweighted transition systems that have recently attracted interest in the context of energy games. We propose a simple fragment of CTL that is able to express properties about accumulated weights along maximal runs of multiweighted modal automata. Our logic is equipped with a game-based semantics and guarantees both soundness (formula satisfaction is propagated to the modal refinements) as well as completeness (formula non-satisfaction is propagated to at least one of its implementations). We augment our theory with a summary of decidability and complexity results of the generalized model checking problem, asking whether a specification—abstracting the whole set of its implementations—satisfies a given formula.

Contributions

- Syntax and game-based semantics for a fragment of CTL reasoning of the accumulated weights of weighted modal transition systems.
- Proof of soundness and completeness of the logic.
- (Un)decidability results for various fragments of the logic.

Publication History

The paper was accepted and presented at the 6th International Symposium on Theoretical Aspects of Software Engineering (TASE'12) and published in Proc. of TASE'12, pages 77-84, IEEE Computer Society Press, 2012.

Part II.

Papers

A Energy Games in Multi-weighted Automata

Uli Fahrenberg
INRIA/IRISA, Rennes Cedex, France

Line Juhl Kim G. Larsen Jiří Srba
Aalborg University, Department of Computer Science, Denmark

Abstract Energy games have recently attracted a lot of attention. These are games played on finite weighted automata and concern the existence of infinite runs subject to boundary constraints on the accumulated weight, allowing e.g. only for behaviours where a resource is always available (nonnegative accumulated weight), yet does not exceed a given maximum capacity. We extend energy games to a multiweighted and parameterized setting, allowing us to model systems with multiple quantitative aspects. We present reductions between Petri nets and multiweighted automata and among different types of multiweighted automata and identify new complexity and (un)decidability results for both one- and two-player games. We also investigate the tractability of an extension of multiweighted energy games in the setting of timed automata.

1. Introduction

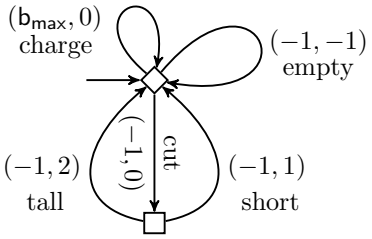
Energy games are two-player games played on finite weighted graphs with the objective of finding an infinite run where the accumulated weight is constrained by a lower and possibly also an upper bound. Such games have attracted considerable attention [BFLM10, BFL+08, BJK10, CdAHS03, Cha10, CDHR10, CD10, DDG+10, EM79, ZP96] in recent years, as they find natural applications in design and analysis of resource-constrained reactive systems, e.g. embedded or hybrid systems.

We study *multiweighted* energy games, where the weight vectors can have an arbitrary dimension. Let us motivate the study by a small example of an automatic lawn mower with a rechargeable battery and a container for collecting grass. Both the battery and the container have a maximum capacity that cannot be exceeded. We assume that the battery can be recharged and the container can be emptied at nearby servicing stations. The charger is an old-fashioned one, and it charges only for a fixed amount of energy corresponding to going from discharged to fully charged. If the lawn mower starts charging while the battery is not fully discharged, the battery will break. The station for emptying the container removes a unit amount of grass at a time and consumes a unit of battery energy. The container will break if too much grass is stored in it.

A weighted game describing the lawn mower behaviour is given in Figure 1a. Each transition has a 2-dimensional vector representing the change to the accumulated battery level in the first coordinate and to the accumulated volume of grass in the container in the second coordinate. The numbers $\mathsf{b}_{\mathsf{max}}$ and $\mathsf{c}_{\mathsf{max}}$ represent the maximum capacity of the battery and the container, respectively. The initial state drawn as a diamond is controlled by Player 1 (the existential player), while the other state drawn as a square is controlled by Player 2 (the universal player).

In the initial state, Player 1 has the choice of either charging the battery, emptying the container or cutting the grass. Moving to the lawn costs one unit of battery energy, and then Player 2 (the environment) controls whether the actual mowing, which costs again one energy unit, will fill the container with one or two units of grass, depending on whether the grass was short or tall. A *configuration* of the game consists of the state and the accumulated weight in all coordinates. A *run* is a sequence of transitions between configurations formed by the players of the game and starting from the initial state with zero accumulated weight.

The question we ask now (the problem called *energy games with lower and upper bounds*) is whether Player 1 has a strategy so that in the infinite run of actions the lawn mower performs, starting with empty battery and empty container, both the accumulated battery level as well as the container content stay invariantly above zero and do not exceed the given upper bounds $\mathsf{b}_{\mathsf{max}} = 4$ and $\mathsf{c}_{\mathsf{max}} = 3$. Such a strategy exists and it is depicted in Figure 1b. Figure 1c illustrates a finite run of the lawn mower game according to this strategy. If we lower the volume of the container to $\mathsf{c}_{\mathsf{max}} = 2$, no such strategy exists. Player 1 must take the charge transition as the first step, after which cutting is the only opportunity. Player 2 can now choose to cut the short grass, leading to battery level 2 and grass volume 1. From here Player 1 can only empty the container, as cutting would allow Player 2 to break the container. After emptying the container,

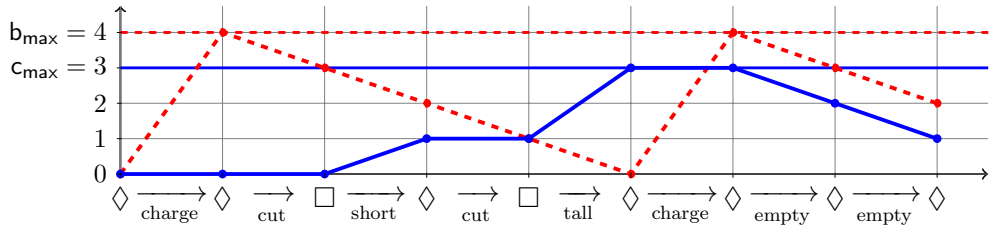


(a) A lawn mower 2-weighted game

$\mathsf{b}_{\max} = 4$ (battery capacity)
$\mathsf{c}_{\max} = 3$ (container capacity)

if *battery* $= 0$ **then** charge
else if *battery* ≥ 2 and *container* ≤ 1 **then** cut
else if *battery* ≥ 1 and *container* ≥ 1 **then** empty

(b) A winning strategy for Player 1



(c) A run of the game (dashed line shows battery level, solid line container content)

Figure 1.: A lawn mower example

battery level is 1 and no transition (apart from cutting) is possible.

There are several variants of the above energy game problem. If we e.g. assume a modern battery charger which does not break the battery when it is not empty, then we have another variant of the problem called *energy games with weak upper bounds*. The weak upper bound game allows taking transitions that will exceed the upper bounds, but these will never accumulate more energy than the maximum capacity. We may also consider infinite runs that are constrained only by a given lower bound but with no upper bound. Finally, we ask questions regarding *parameterization*. We want to decide whether there *exists* some battery capacity $\mathsf{b}_{\max}$ and some initial battery level such that Player 1 wins the energy game with lower and upper bound (or some of its variants). In our example one can by a simple reasoning argue that for a container capacity $\mathsf{c}_{\max} = 2$, there is no battery capacity $\mathsf{b}_{\max}$ so that Player 1 can guarantee an infinite behaviour of the lawn mower.

Contributions. We define the variants of multiweighted energy games (Section 2) and present reductions involving these games, leading to new decidability and complexity results. Some reductions are to/from Petri nets (Section 3) while others are between different multiweighted energy games (Section 4). This is followed by a summary of decidability and complexity results we achieved. In Section 6 we consider a parameterized version of existential one-player games and show that some variants of the problem lead to undecidability while others are decidable in polynomial time. We conclude by presenting an undecidability result for a natural timed extension of the energy games (Section 7).

Related work. The idea of checking whether a given resource stays above zero at all times was first presented by Chakrabarti et al. in [CdAHS03], treating the subject in relation to interfaces. The lower and (weak) upper bound problems were first formulated in [BFL+08] for the case with a single weight. The paper presents several complexity results for the 1-weighted case, both timed and untimed, and has given rise to a number of recent papers on 1-weighted energy games [CDHR10, CD10, DDG+10].

The multiweighted extension has been studied in [BJK10], but only for energy games with *unary* weights, i.e. updates by 1, 0 or −1. A continuation of this work presents a polynomial time algorithm for the 2-weighted case with unary inputs [Cha10]. Contrary to this line of work, we consider *binary* input encoding, hence weight updates are now drawn from the full set of integers. Also in contrast to [BJK10, Cha10], where only complexity *upper* bounds are given, we give complexity *lower* bounds that in most cases match the upper bounds.

Multiweighted energy games with general integer updates have been considered in [CDHR10], where the authors show that the problem of deciding the existence of an initial weight vector such that Player 1 can win the lower bound energy game is solvable in polynomial time. In contrast to this, we show here that the non-parameterized variant of this problem—can Player 1 win with a *given* initial weight vector—is EXPSPACE-hard. We also treat the parameterized setting, where we show that the existential lower and (weak) upper bound problems with both bounds and initial weight vector parameterized are also decidable in polynomial time, unless the upper bound parameter is used in the transitions of the automaton, in which case the problem becomes undecidable.

2. Multiweighted Automata and Games

We denote by $\mathbb{Z}^k$ the set of integer vectors of dimension $k > 0$ and by $\bar{w}[i]$ the i'th coordinate of a vector $\bar{w} \in \mathbb{Z}^k$. A *k-weighted game* G is a four-tuple $(Q_1, Q_2, q_0, \longrightarrow)$ where Q_1 and Q_2 are finite, disjoint sets of *existential* and *universal* states, respectively, $q_0 \in Q_1 \cup Q_2$ is the initial state and $\longrightarrow \subseteq (Q_1 \cup Q_2) \times \mathbb{Z}^k \times (Q_1 \cup Q_2)$ is a finite weighted transition relation, written as $q \xrightarrow{\bar{w}} q'$ whenever $(q, \bar{w}, q') \in \longrightarrow$. We refer to Figure 1a in the introduction for an example of a k-weighted game with $k = 2$.

We are interested only in infinite runs in multiweighted games, hence for the rest of the paper, we assume that the game G is non-blocking, i.e. for every $q \in Q_1 \cup Q_2$ we have $q \xrightarrow{\bar{w}} q'$ for some $\bar{w} \in \mathbb{Z}^k$ and $q' \in Q_1 \cup Q_2$.

A *weighted run* in a k-weighted game $G = (Q_1, Q_2, q_0, \longrightarrow)$ restricted to a *weak upper bound* $\bar{b} \in (\mathbb{N}_0 \cup \infty)^k$ is an infinite sequence $(q_0, \bar{v}_0), (q_1, \bar{v}_1), (q_2, \bar{v}_2), \ldots$ where $q_0, q_1, \ldots \in Q_1 \cup Q_2$, $\bar{v}_0 = \bar{0} = (0, 0, \ldots, 0)$ and $\bar{v}_1, \bar{v}_2, \ldots \in \mathbb{Z}^k$ such that for all $j \geq 0$ we have $q_j \xrightarrow{\bar{w}_j} q_{j+1}$ and

$$\bar{v}_{j+1}[i] = \min\left\{\bar{v}_j[i] + \bar{w}_j[i], \bar{b}[i]\right\}$$

for all coordinates i. An illustration of a run in a 2-weighted game is given in Figure 1c in the introduction. Intuitively, a weighted run is a sequence of states together with the accumulated weight gathered along the path. Moreover, the accumulated weight is

truncated, should it exceed in some coordinate the given maximum weight $\bar{b}$. By $\mathsf{WR}_{\bar{b}}(G)$ we shall denote the set of all weighted runs in G restricted to the maximum accumulated weight $\bar{b}$.

A *strategy* for Player $i \in \{1,2\}$ in a k-weighted game $G = (Q_1, Q_2, q_0, \longrightarrow)$ (restricted to a weak upper bound $\bar{b}$) is a mapping σ from each finite prefix of a weighted run in $\mathsf{WR}_{\bar{b}}(G)$ of the form $(q_0, \bar{v}_0), \ldots, (q_n, \bar{v}_n)$ with $q_n \in Q_i$ to a configuration $(q_{n+1}, \bar{v}_{n+1})$ such that $(q_0, \bar{v}_0), \ldots, (q_n, \bar{v}_n), (q_{n+1}, \bar{v}_{n+1})$ is a prefix of some weighted run in $\mathsf{WR}_{\bar{b}}(G)$. A weighted run $(q_0, \bar{v}_0), (q_1, \bar{v}_1), \ldots$ *respects* a strategy σ of Player i if $\sigma((q_0, \bar{v}_0), \ldots, (q_n, \bar{v}_n)) = (q_{n+1}, \bar{v}_{n+1})$ for all n such that $q_n \in Q_i$. Figure 1b in the introduction shows a strategy for the 2-weighted game from Figure 1a; note that the run of the game depicted in Figure 1c indeed respects this strategy.

We shall consider three decision problems related to energy games on a given k-weighted game $G = (Q_1, Q_2, q_0, \longrightarrow)$. Below we let $\overline{\infty} = (\infty, \infty, \ldots, \infty)$, and we write $\bar{w} \leq \bar{v}$ if $\bar{w}[i] \leq \bar{v}[i]$ for all i, $1 \leq i \leq k$.

Energy Game with Lower bound (GL): Given a game G, is there a strategy σ for Player 1 such that any weighted run $(q_0, \bar{v}_0), (q_1, \bar{v}_1), \ldots \in \mathsf{WR}_{\overline{\infty}}(G)$ respecting σ satisfies $\bar{0} \leq \bar{v}_i$ for all $i \geq 0$?

Hence we ask whether Player 1 has a winning strategy such that during any play the accumulated weight stays above zero in all coordinates.

Energy Game with Lower and Weak upper bound (GLW): Given a game G and a vector of upper bounds $\bar{b} \in \mathbb{N}_0^k$, is there a strategy σ for Player 1 such that any weighted run $(q_0, \bar{v}_0), (q_1, \bar{v}_1), \ldots \in \mathsf{WR}_{\bar{b}}(G)$ respecting σ satisfies $\bar{0} \leq \bar{v}_i$ for all $i \geq 0$?

Hence we ask whether Player 1 has a winning strategy such that during any play the accumulated weight, which is truncated whenever it exceeds the given upper bound, stays in all coordinates above zero.

Energy Game with Lower and Upper bound (GLU): Given a game G and a vector of upper bounds $\bar{b} \in \mathbb{N}_0^k$, is there a strategy σ for Player 1 such that any weighted run $(q_0, \bar{v}_0), (q_1, \bar{v}_1), \ldots \in \mathsf{WR}_{\overline{\infty}}(G)$ respecting σ satisfies $\bar{0} \leq \bar{v}_i \leq \bar{b}$ for all i?

Hence we ask whether Player 1 has a winning strategy such that during any play the accumulated weight stays in all coordinates above zero and below the given upper bound.

The problems GL, GLW and GLU can be specialized in two different ways. Either by giving Player 1 the full control over the game by setting $Q_2 = \emptyset$ or dually by giving the full control to Player 2 by assuming that $Q_1 = \emptyset$. The first problem is called the *existential variant* as we essentially ask whether there *exists* some weighted run with the accumulated weight within the given bounds. The second problem is called the *universal variant* as we now require that *all* weighted runs satisfy the constraints of the energy game. We will denote the respective existential problems by EL, ELW and ELU, and the universal problems by AL, ALW and ALU. These special cases are known as one-player

games or simply as multiweighted automata, and we denote such games as only a triple $(Q, q_0, \longrightarrow)$.

In the general formulation of energy games there is no fixed bound on the dimension of the weight vectors, in other words, the dimension k is a part of the input. If we want to consider problems of a fixed dimension k, we use the notation $\mathrm{GL}(k)$, $\mathrm{GLW}(k)$, $\mathrm{GLU}(k)$, $\mathrm{EL}(k)$ etc.

As the inputs to our decision problems are numbers, it is important to agree on their encoding. We will use the *binary encoding*, unlike some other recent work [BJK10, Cha10] where unary notation is considered and thus enables to achieve better complexity bounds as the size of their input instance is exponentially larger.

We may also easily allow an initial weight vector $\bar{w}_0$ different from $\bar{0}$. This is evident by adding a new fresh start state with one transition labeled with $\bar{w}_0$ pointing to the original start state. In addition we may assume that in any given upper bound or weak upper bound vector $\bar{b}$ we have $\bar{b}[1] = \bar{b}[2] = \cdots = \bar{b}[k]$. This can be achieved by scaling every i'th coordinate of all weight vectors on transitions with $\frac{\bar{b}[1]\cdot\ldots\cdot\bar{b}[k]}{\bar{b}[i]}$ in order to obtain equality on the coordinates of $\bar{b}$. Such a scaling implies only polynomial increase in the size (in binary encoding) of the upper bound constants.

3. Relationship to Petri Nets

We show that the existential variants of the infinite run problems on multiweighted automata can be reduced to the corresponding problems on Petri nets and vice versa. This will allow us to transfer some of the decidability and complexity results from the Petri net theory to our setting.

We shall first define the Petri net model with weighted arcs (that allow to consume more than one token from a given place). A *Petri net* is a triple $N = (P, T, W)$ where P is a finite set of places, T is a finite set of transitions, and $W : (P \times T) \cup (T \times P) \to \mathbb{N}_0$ is a function assigning a weight to each arc in the net. A *marking* on N is a function $M : P \to \mathbb{N}_0$ denoting the number of tokens present in the places. A *marked Petri net* is a pair (N, M_0) where N is a Petri net and M_0 is an initial marking on N.

A transition $t \in T$ is *enabled* in a marking M if $M(p) \geq W(p,t)$ for all $p \in P$. An enabled transition may *fire*. When a transition t fires, it produces a new marking M' obtained as $M'(p) = M(p) - W(p,t) + W(t,p)$ for all places $p \in P$. Then we write $M \xrightarrow{t} M'$. A marking M is *reachable* in N if $M_0 \longrightarrow^* M$ where $\longrightarrow = \bigcup_{t \in T} \xrightarrow{t}$. A marked Petri net is called *1-safe* if for any reachable marking M the number of tokens in any place is at most one, i.e. $M(p) \leq 1$ for all $p \in P$. We say that a marked net (N, M_0) has an infinite run if there is a sequence of markings $M_1, M_2, \ldots$ and transitions $t_1, t_2, \ldots$ such that $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \xrightarrow{t_3} \ldots$ The *infinite run problem* for Petri nets (see e.g. [Esp98]) is to decide whether a given Petri net has an infinite run.

Lemma 1. *The infinite run Petri net problem is polynomial time reducible to EL. The infinite run Petri net problem for 1-safe nets is polynomial time reducible to ELU and*

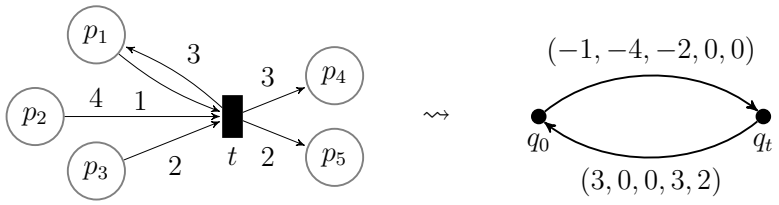


Figure 2.: Translation of a Petri net to a 5-weighted automaton

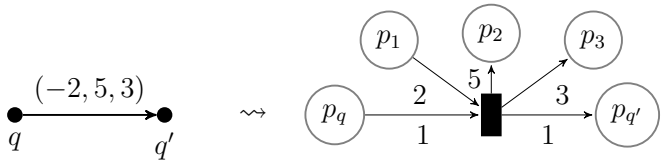


Figure 3.: Translation of a 3-weighted automaton to a Petri net

ELW. The problem EL is polynomial time reducible to the infinite run problem of Petri nets.

Proof. We first prove the first part of the lemma. Given a Petri net $N = (P, T, W)$ where $P = \{p_1, \ldots, p_k\}$ we construct a k-weighted automaton $A = (Q, q_0, \longrightarrow)$ such that $Q = \{q_0\} \cup \{q_t \mid t \in T\}$. Now for every $t \in T$ we add to A two transitions $q_0 \xrightarrow{\bar{w}_t^-} q_t$ and $q_t \xrightarrow{\bar{w}_t^+} q_0$ where $\bar{w}_t^-[i] = -W(p_i, t)$ and $\bar{w}_t^+[i] = W(t, p_i)$ for all i, $1 \leq i \leq k$. Consult Figure 2 for an example. The initial weight vector then corresponds to the initial marking of the net in the expected way. It follows from the construction that each transition firing can be simulated by two transitions in the constructed weighted automaton and vice versa. Observe that the reachable Petri net markings are represented as accumulated weight vectors in the automaton and hence are nonnegative in all coordinates. It is easy to verify that the net has an infinite run if and only if the EL problem has a solution. The reduction clearly runs in polynomial time.

For the second part, observe that if the net is 1-safe then by taking the upper bound $\bar{b} = (1, 1, \ldots, 1)$ we have a reduction from the infinite run problem for 1-safe nets to ELU and ELW.

The reduction from k-weighted automata to Petri nets works in a similar way. Given a k-weighted automaton $A = (Q, q_0, \longrightarrow)$ we construct a Petri net $N = (P, T, W)$ where $P = \{p_1, \ldots, p_k\} \cup \{p_q \mid q \in Q\}$ and $T = \{t_{(q,\bar{w},q')} \mid q \xrightarrow{\bar{w}} q'\}$. For each $t_{(q,\bar{w},q')}$ we set $W(p_q, t_{(q,\bar{w},q')}) = 1$, $W(t_{(q,\bar{w},q')}, p_{q'}) = 1$ and for all i, $1 \leq i \leq k$, $W(p_i, t_{(q,\bar{w},q')}) = -\bar{w}[i]$ if $\bar{w}[i] < 0$ and $W(t_{(q,\bar{w},q')}, p_i) = \bar{w}[i]$ if $\bar{w}[i] \geq 0$. See Figure 3 for an example of the reduction. The initial marking corresponds to the initial weight vector in the natural way, and there is one extra token in the place p_{q_0} representing the current state of the automaton. As before, it is easy to verify that the constructed Petri net has an infinite run if and only if the EL problem has a solution. The reduction clearly runs in polynomial time. ■

Theorem 2. *The problem EL is EXPSPACE-complete. The problems ELU and ELW are PSPACE-complete.*

Proof. The complexity bounds for EL follow from Lemma 1 and from the fact that the existence of an infinite run in a Petri net is decidable in EXPSPACE [Yen92, AH09] and EXPSPACE-hard (see e.g. [Esp98]). The same problem for 1-safe Petri nets is PSPACE-complete (see again [Esp98]) and by Lemma 1 we get PSPACE-hardness also for ELU and ELW. The containment of the ELU and ELW problems in PSPACE can be shown by noticing that these problems have an infinite run $(q_0, \bar{v}_0), (q_1, \bar{v}_1), \ldots$ if and only if there are two indices $i < j$ such that $(q_i, \bar{v}_i) = (q_j, \bar{v}_j)$. As the size of any configuration $(q, \bar{v})$ appearing on such a run is polynomially bounded by the size of the input (which includes the upper bound vector), we can use a nondeterministic algorithm to guess such a repeated configuration $(q_i, \bar{v}_i)$ and nondeterministically verify whether it forms a loop which is reachable from the initial pair $(q_0, \bar{v}_0)$. This completes the argument for the containment of ELU and ELW in PSPACE. ■

4. Reductions among Energy Games

In this section we present reductions among the variants of one- and two-player energy games with a particular focus on the size of the weight vectors.

Theorem 3. *The problem GLU(k) is polynomial time reducible to GL($2k$) and GLW($2k$) for all $k > 0$. The reduction preserves the existential and universal variants of the problems.*

Proof. Let $G_k = (Q_1, Q_2, q_0, \longrightarrow)$ be a k-weighted game and let $\bar{b}$ be a given upper bound vector for the GLU problem. We construct a $2k$-weighted game $G_{2k} = (Q_1 \uplus \{q_s\}, Q_2, q_s, \longrightarrow)$ where $q \xrightarrow{(\bar{w}[1],\bar{w}[2],\ldots,\bar{w}[k],-\bar{w}[1],-\bar{w}[2],\ldots,-\bar{w}[k])} q'$ in G_{2k} if and only if $q \xrightarrow{\bar{w}} q'$ in G_k. We moreover add the initial transition $q_s \xrightarrow{\bar{w}_0} q_0$ where $\bar{w}_0[i] = 0$ and $\bar{w}_0[k+i] = \bar{b}[i]$ for all i, $1 \leq i \leq k$. Figure 4 illustrates the construction on an example. Intuitively, every coordinate in the weight vector is duplicated and the duplicated coordinate gets initially the value from the vector $\bar{b}$, while the original coordinate is 0. It is now easy to verify that during any run in G_{2k} all its configurations $(q, \bar{v})$ satisfy the invariant $\bar{v}[i] + \bar{v}[k+i] = \bar{b}[i]$ for all i, $1 \leq i \leq k$.

The upper bound check is hence replaced with a lower bound on the duplicate coordinates and hence the GLU problem is reduced to GL and also to GLW (by using the weak upper bound vector $\bar{b}$), while the size of the weight vectors doubles. The reduction also clearly preserves the existential and universal variants of the problems. ■

Since we already know that ELU(1) is NP-hard [BFL+08], using Theorem 3 with $k = 1$ gives that EL(2) is NP-hard too, which is of course then also the case for EL. Similarly as GLU(1) is known to be EXPTIME-hard [BFL+08], we get EXPTIME-hardness also for GL(2) and hence also for GL.

Our next reductions show (perhaps surprisingly) that allowing multiple weights is not that crucial in terms of complexity. The first theorem shows that for upper bound games,

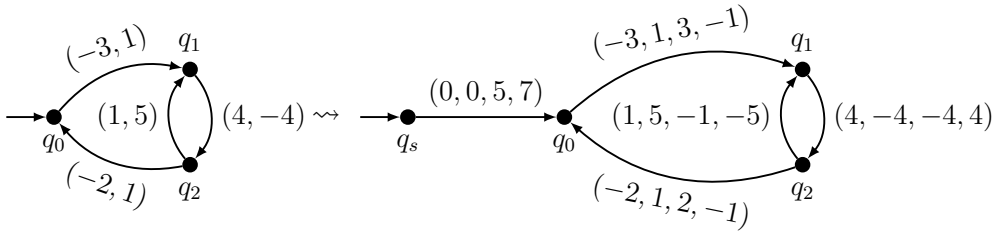


Figure 4.: Example of reduction from GLU with $\bar{b} = (5, 7)$ to GL

it suffices to work with one weight only; Theorem 5 then shows that for the existential variant, two weights are enough.

Theorem 4. *The problem GLU is polynomial time reducible to GLU*(1).

Proof. Let $G = (Q_1, Q_2, q_0, \longrightarrow)$ be a k-weighted game and $\bar{b}$ a given upper bound vector for the GLU problem. We assume that G is encoded in binary and let n denote the size of such encoding. This means that all constants that appear in the description of G are less than 2^n. We will construct a corresponding 1-weighted game $G' = (Q'_1, Q'_2, q_s, \longrightarrow)$ where $Q'_1 = Q_1 \cup \{q_2, q_3, \ldots, q_{k+5} \mid q \in Q_1\} \cup \{q_s\}$ and $Q'_2 = Q_2 \cup \{q_1 \mid q \in Q_2\}$ that simulates G.

Let $\bar{w}$ denote any weight vector present in G. Clearly, $0 \leq \bar{w}[1], \ldots, \bar{w}[k] < 2^n$ due to the encoding of the input. Without loss of generality we can assume that all coordinates of $\bar{b}$ are the same, i.e. that $\bar{b} = (b, \ldots, b)$ for some $0 \leq b < 2^n$.

We need to encode the weights from G using only one weight. We will do so by placing them into the single (large) weight w'. Since $b < 2^n$, at most n bits are needed to represent each weight $\bar{w}[i]$. The weight w' is constructed by appending the weights from G in higher and higher bit positions, with a suitable separation sequence to ensure that weights cannot get 'entangled' should their bounds overflow or underflow. Formally, we introduce the following notation for any integer $\ell \in \mathbb{Z}$ and any i, $1 \leq i \leq k$:

$$\langle \ell \rangle^i = \ell \cdot 2^{(i-1)(n+2)} \ .$$

For example, if $n = 4$ then $\langle 6 \rangle^2 = 6 \cdot 2^6 =$ (in binary) $= 110 \cdot 1000000 = 110000000$. A weight vector $\bar{w}$ of size k in G is now represented by the number

$$\langle \bar{w} \rangle \stackrel{\text{def}}{=} \langle \bar{w}[1] \rangle^1 + \langle 2^{n+1} \rangle^1 + \langle \bar{w}[2] \rangle^2 + \langle 2^{n+1} \rangle^2 + \ldots + \langle \bar{w}[k] \rangle^k + \langle 2^{n+1} \rangle^k$$

where the weights $\bar{w}[1], \ldots, \bar{w}[k]$ written in binary from the less significant bits to more significant ones are separated by the binary string '10'. For example if again $n = 4$ then the weight vector $\bar{w} = (110, 1, 1011)$ with the weights written in binary is represented by the binary number $\langle \bar{w} \rangle = 10\ 1011\ 10\ 0001\ 10\ 0110$.

The new upper bound B for G' is defined by $B = \langle b \rangle^{k+1} + \langle \bar{b} \rangle$ where apart from the standard encoding of all upper bounds for all coordinates we add one more time the constant b at the most significant bits (we will use these bits for counting in our construction).

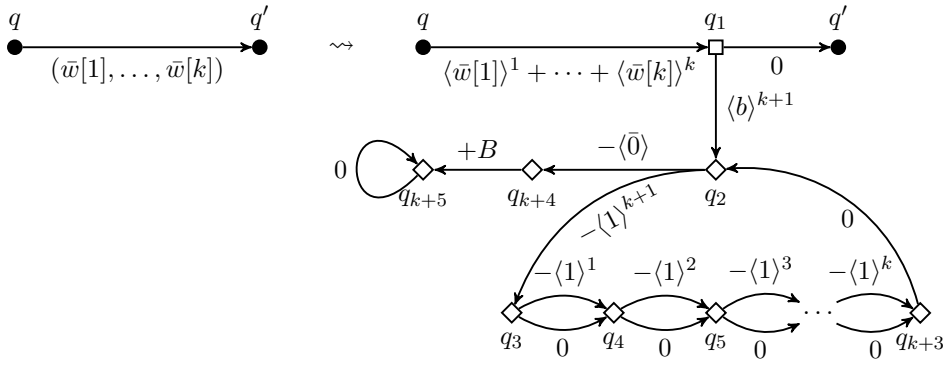


Figure 5.: Simulation of a transition in a k-weighted game by a 1-weighted game

Each transition $q \xrightarrow{\bar{w}} q'$ in G is transformed into a number of transitions in G' as depicted in Figure 5 where Player 1 (existential) states are drawn as diamonds and Player 2 (universal) states are drawn as squares. The states drawn as filled circles can be of either type, and their type is preserved in the translation. We also add the initial transition $q_s \xrightarrow{\langle \bar{0} \rangle} q_0$ which inserts the separation strings 10 at the correct positions.

The idea is that the update of the accumulated weight vector $\bar{v}$ in G via adding a vector $\bar{w}$ like in Figure 5 is simulated by adding the numbers $\langle \bar{w}[1] \rangle^1$, $\langle \bar{w}[2] \rangle^2$, ..., $\langle \bar{w}[k] \rangle^k$ to the accumulated weight in G'. The chosen encoding of k weights into a single weight is crucial to preserve the soundness of the construction as discussed in the following remark.

Remark 1. Given an accumulated weight vector $\bar{v}$ and a weight update vector $\bar{w}$ where $\bar{0} \leq \bar{v}, \bar{w} \leq \bar{b} < (2^n, \ldots, 2^n)$, then adding the numbers $\langle \bar{v} \rangle$ and $\langle \bar{w}[i] \rangle^i$ in $\langle \bar{v} \rangle$ changes at most the bits that are designated for representing the weight coordinate $\bar{w}[i]$ and the separating two bits 10 just before it. This can be easily seen by analyzing the two extreme cases of adding $11 \ldots 1$ to an accumulated weight coordinate with full capacity and subtracting $11 \ldots 1$ from an accumulated weight coordinate that represents zero as showed in the following two examples.

$$\begin{array}{r} \ldots 10\ 111 \ldots 111\ 10 \ldots \\ +\ldots 00\ 111 \ldots 111\ 00 \ldots \\ \hline \ldots 11\ 111 \ldots 110\ 10 \ldots \end{array} \qquad \begin{array}{r} \ldots 10\ 000 \ldots 000\ 10 \ldots \\ -\ldots 00\ 111 \ldots 111\ 00 \ldots \\ \hline \ldots 01\ 000 \ldots 001\ 10 \ldots \end{array}$$

Let us now argue about the correctness of this polynomial time construction. Assume that Player 1 has a winning strategy in the game G. As the accumulated weight stays within the bounds during any such play in G, it is clear that the same winning strategy can be performed also in G' using only a single weight. One complication is that each transition in G is split in G' and a new node for Player 2 (q_1 in Figure 5) is inserted. Hence Player 2 could possibly have an extra winning strategy by playing $q_1 \xrightarrow{\langle b \rangle^{k+1}} q_2$, instead of the expected move to q'. However, because the accumulated weight vector $\bar{v}$ satisfies $0 \leq \bar{v}[i] \leq b < 2^n$ for all i, we can see that Player 1 wins in this case, by taking

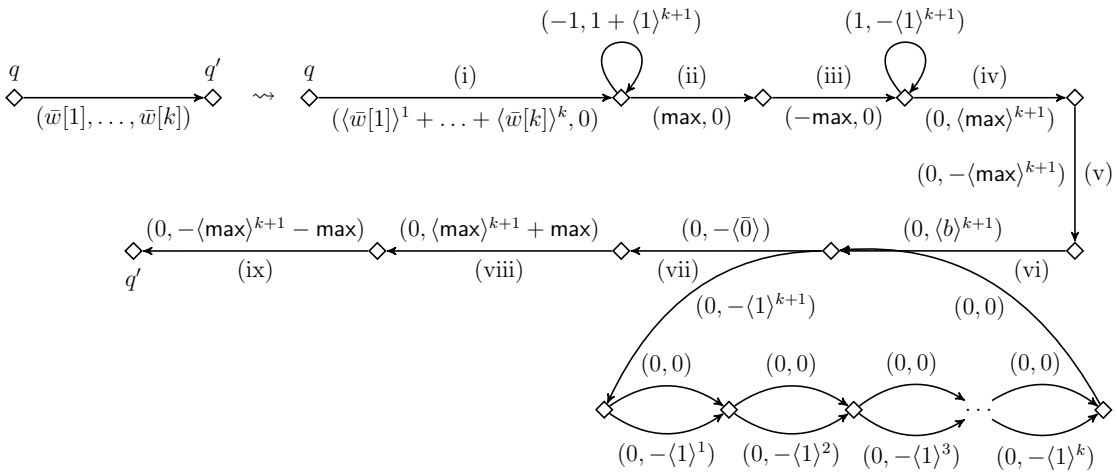


Figure 6.: Simulation of a k-weighted transition by a 2-weighted automaton

the loop $q_2, q_3, \ldots q_{k+3}, q_2$ exactly b times while choosing the zero or $-\langle 1\rangle^i$ transitions (for all i) in such a way that the bits representing the weight $\bar{v}[i]$ are all set to 0. What remains in G' as the accumulated weight is then the value $\langle \bar{0}\rangle$ which consists only of the separation symbols. From here Player 1 takes the transition with weight $-\langle \bar{0}\rangle$, setting the accumulated weight to zero, and wins by performing the transition labeled with $+B$ (which is possible only if the accumulated weight is exactly zero) and repeatedly performing in q_{k+5} the self-loop with weight zero.

On the other hand, assume that a play in G causes the accumulated weight in some coordinate i, $1 \leq i \leq k$, to get out of the bounds; we shall argue that Player 2 has a winning strategy in G' in this case. Should this happen during a transition from q to q' in G, then in G', Player 2 will simply move from the intermediate state q_1 to q_2, while the counter value of size b is added to the most significant bits of the accumulated weight via adding the number $\langle b\rangle^{k+1}$. It is clear that it is possible to move from q_2 to q_{k+5} only if the accumulated weight is exactly $\langle \bar{0}\rangle$. In order to achieve this value, the accumulated weight needs to be decreased exactly b times via taking the loop $q_2, q_3, \ldots, q_{k+3}, q_2$. Because of Remark 1 we can see that only the bits relevant to each weight coordinate were changed before entering the loop, so it is impossible to zero all bits corresponding to the coordinate i while preserving the separation bits 10. ■

Theorem 5. *The problem ELU is polynomial time reducible to ELU*(2).

Proof. The reduction idea is similar to the one in the proof of Theorem 4. The main complication is that Player 2 has no states in control, hence checking the underflow and overflow of weights has to be performed without resorting to an opponent. As the original weight values are destroyed during such a check, we need to employ a second weight for saving them.

Let $A = (Q, q_0, \longrightarrow)$ be a k-weighted automaton and $\bar{b}$ the upper bound vector for the ELU problem. We construct a corresponding 2-weighted automaton $A' = (Q', q_s, \longrightarrow)$.

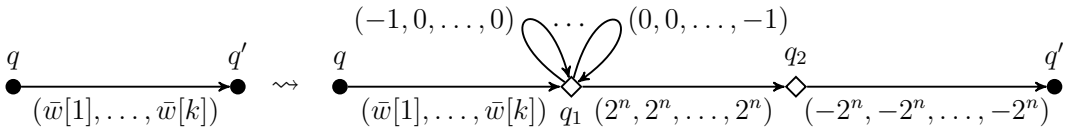


Figure 7.: Simulation of a transition in a LW game by a LU game

Let $\bar{w}$ denote the weight vectors in A and $\bar{v}[1], \bar{v}[2]$ the two weights in A'. As before for an input automaton of size n we may assume that all weights in A have the same upper bound $\bar{b} = (b, b, \ldots, b)$ where $b < 2^n$.

The upper bound $\bar{b}'$ for the ELU(2) problem in A' is given by $\bar{b}' = (\mathsf{max}, \langle \mathsf{max} \rangle^{k+1} + \mathsf{max})$ with $\mathsf{max} = \langle b \rangle^{k+1} + \langle \bar{b} \rangle$. The reason for reserving twice as many bits in the second weight is that we need to save there *two* copies of the first weight. Figure 6 shows how to simulate one transition in A by a number of transitions in A'. From the newly added initial state q_s we also add the transition $q_s \xrightarrow{(\langle \bar{0} \rangle, 0)} q_0$ which inserts the separation strings '10' into the first weight.

We shall now argue that the automaton A' faithfully simulates A. We will examine the effect of the sequence of transitions between q and q' added to the automaton A' (here numbered with (i), (ii), ..., (ix) for convenience) and argue at the same time that the part of the run between q and q' in A' is uniquely determined. By construction, $\bar{v}[2]$ will be zero when entering q, and then the transition (i) adds the encoded weights of the original transition in A to $\bar{v}[1]$. Transition (ii) will add the upper bound to $\bar{v}[1]$, hence before this, we need to take the loop with weight $(-1, 1 + \langle 1 \rangle^{k+1})$ until $\bar{v}[1]$ equals zero, thereby copying twice the value of $\bar{v}[1]$ to $\bar{v}[2]$ (first copy in the less significant bits, second copy in the more significant bits). After the transitions (ii) and (iii), $\bar{v}[1]$ is then again at zero. Now transition (iv) wants to add the upper bound to the most significant bits of $\bar{v}[2]$, hence before this, we need to take the loop with weight $(1, -\langle 1 \rangle^{k+1})$ until the value of the most significant bits in $\bar{v}[2]$ is copied to $\bar{v}[1]$, thereby restoring the original weight in $\bar{v}[1]$.

After the transitions (iv) and (v), we are in a situation where both coordinates in the accumulated weight store the same number, and we can afford to destroy the second copy during the verification phase for bound overflow/underflow performed by transitions (vi), the long loop, and transitions (vii), (viii) and (ix). This is identical to the construction in the previous proof (except for the extra coordinate $\bar{v}[1]$ which is not updated). Provided that no violation of bounds was detected, we will reach q' with $\bar{v}[1]$ encoding the weight vector of A at q' and $\bar{v}[2]$ equal to zero.

Hence a transition between two states in A can be performed if and only if the sequence of transitions between q and q' in A' can be performed. As the reduction is clearly in polynomial time, this concludes the proof. ■

The next theorem finishes our considerations about reductions between different variants of energy games.

Theorem 6. *The problem GLW is polynomial time reducible to GLU, and ELW is polynomial time reducible to ELU.*

Proof. Let $G = (Q_1, Q_2, q_0, \longrightarrow)$ be a k-weighted game and let $\bar{b}$ be a given upper bound vector for the GLW problem. We will construct a corresponding k-weighted game $G' = (Q'_1, Q'_2, q_0, \longrightarrow)$ where $Q'_1 = Q_1 \cup \{q_1, q_2 \mid q \in Q_1\}$ and $Q'_2 = Q_2$ that simulates G.

As before we assume that the weak upper bound is $\bar{b} = (b, \ldots, b)$ and that b is represented using at most n bits, hence $0 \leq b < 2^n$. The new upper bound for G' is given as $\bar{b}' = (b', \ldots, b')$ where $b' = 2^n + b$ (in binary the most significant bit 1 is appended to the binary encoding of b).

Each transition $q \xrightarrow{\bar{w}} q'$ in G is simulated by a number of transitions in G' as seen in Figure 7. Moving from q to q_1 adds $\bar{w}$ to the accumulated weights of G' in exactly the same way as in G. In q_1 Player 1 has the opportunity to decrement independently all weight coordinates with an arbitrary value. The two last transitions from q_1 to q' make sure that in all coordinates all weights are no more than b, otherwise the upper bound $\bar{b}'$ is exceeded.

It is now clear that if Player 1 has a winning strategy in G, then it has a winning strategy also in G' by lowering all weights above b to exactly b in the state q_1. On the other hand, if Player 1 does not have a winning strategy in G, then it cannot win in G' either. This can be observed by the fact that Player 1 is forced to decrement all weights to at least b, and the player cannot benefit from decrementing them to any lower number as this makes the position of Player 1 in the weak upper bound game only worse.

Since the reduction is clearly in polynomial time and it adds only existential (Player 1) states, this concludes the proof. ■

Now, in combination with Theorems 4 and 5, we get the following corollary.

Corollary 7. *The problems GLW and ELW are polynomial time reducible to GLU*(1) *and ELU*(2)*, respectively.*

5. Summary of Complexity Results

The collection of complexity results and reductions between different types of energy games and automata enables us to draw the conclusions presented in Table 1. Notice that the LU problems are computationally easier than the L problems for an arbitrary number of weights, even though they are harder than the L problems in the 1-weighted case. The configuration space for the LU (and LW) problems is bounded (see Theorem 2), whereas the same a priori does not apply to the L problem.

Observe also that any *universal* problem with k weights can be solved by checking the same problem for each coordinate independently. If the k-weighted problem violates the bounds at some coordinate, so will do the 1-weighted problem projected on this coordinate. On the other hand, if some coordinate in the 1-weighted problem violates the bounds then so will do the k-weighted game, as the same run leading to the violation

Weights	Type	Existential	Game
One	L	$\in$ P [BFL+08]	$\in$ UP $\cap$ coUP [BFL+08]
	LW	$\in$ P [BFL+08]	$\in$ NP $\cap$ coNP [BFL+08]
	LU	NP-hard [BFL+08], $\in$ PSPACE [BFL+08]	EXPTIME-complete [BFL+08]
Fixed (k>1)	L	**NP-hard**, $\in$ k-EXPTIME [BJK10] (Rem. 2)	**EXPTIME-hard**, $\in$ k-EXPTIME [BJK10] (Rem. 3)
	LW	**NP-hard**, $\in$ **PSPACE** (Rem. 4) **PSPACE-complete** for $k \geq 4$	**EXPTIME-complete** (Rem. 5)
	LU	**PSPACE-complete** (Rem. 4)	**EXPTIME-complete** (Rem. 5)
Arbitrary	L	**EXPSPACE-complete** (Thm. 2)	**EXPSPACE-hard** (from EL) decidable [BJK10]
	LW	**PSPACE-complete** (Thm. 2)	**EXPTIME-complete** (Rem. 5)
	LU	**PSPACE-complete** (Thm. 2)	**EXPTIME-complete** (Rem. 5)

Table 1.: Complexity bounds; results obtained in this paper are in bold

in one coordinate leads to a violation in the k-weighted game (unless the violation occurs in some other coordinate before that). As AL(1), ALW(1) and ALU(1) are decidable in P [BFL+08], this implies polynomial upper bounds also for all the other k-weighted universal problems.

Remark 2. The problem ELU(1) is NP-hard, and Theorem 3 implies NP-hardness for EL(2). The upper bound follows from the game version of the problem (see also Remark 3).

Remark 3. The lower bound follows from EXPTIME-hardness of GLU(1) and Theorem 3. The upper bound is due to a result in [BJK10] showing $(k-1)$-EXPTIME containment for GL(k) but for games where weight updates are only $+1$, 0, and -1. We can reduce updates with arbitrary weights into this setting by standard techniques (introducing intermediate transitions which repeatedly add or subtract 1) but this causes an exponential blowup in the size of the system. Hence the complexity upper bound increases by one exponent to k-EXPTIME.

Remark 4. The PSPACE upper bound follows from the results for an arbitrary number of weights (Theorem 2). The PSPACE lower bound for ELU(2) is due to the reduction in Theorem 5 and PSPACE-hardness of ELU. By using Theorem 3 we get PSPACE-hardness for ELW(4) because ELU(2) is PSPACE-hard, and we also get NP-hardness of ELW(2) as ELU(1) is NP-hard.

Remark 5. The upper bound for GLU follows from Theorem 4 and the EXPTIME upper bound for GLU(1); the upper bound for GLW follows additionally from Theorem 6. The lower bound for GLU is obvious and for GLW it is by Theorem 3 and the EXPTIME-hardness result for GLU(1).

6. Parameterized Existential Problems

In this section we shall focus in more detail on the existential one-player energy games. So far we have studied decision problems where both the initial weight vector and the upper bound were given as a part of the input. We will now consider parameterized versions of the problems where, given a weighted automaton, we ask whether there is some initial weight vector $\bar{v}_0$ (and some upper bound $\bar{b}$ in case of ELU and ELW) such that the automaton has a run where the accumulated weight satisfies the constraints imposed by the respective variant of the problem.

Recent work by Chatterjee et al. [CDHR10] proves that the parameterized version of the EL problem, asking if there is an initial weight vector such that the accumulated weight of some run in the automaton stays (component-wise) above zero, is decidable in polynomial time. Perhaps surprisingly, this result contrasts with our EXPSPACE-hardness result for the EL problem where the initial weight vector is fixed. An interesting fact, using Lemma 1, is that by the result of [CDHR10], it is also decidable in polynomial time whether there is an initial marking such that a given Petri net has an infinite run.

The situation can be, however, different when considering the problems ELU and ELW. Depending on whether the parameterized upper bound $\bar{b}$ is allowed to appear as a weight in transitions of the given weighted automaton (see Section 1 for an example where the upper bound appears as a weight), we shall show below that the problem is either decidable in polynomial time or undecidable.

We present first the positive result. Its proof is based on a polynomial time algorithm for zero-weight cycle detection in multiweighted automata by Kosaraju and Sullivan [KS88], and we acknowledge [CDHR10] where we found a pointer to this result, which is mentioned there in connection with the parameterized EL problem.

Theorem 8. *The parameterized ELU and ELW problems where the upper bound parameter does not appear as a weight in the underlying weighted automaton are decidable in polynomial time.*

Proof. We shall first focus on the ELU problem. Notice that a parameterized ELU problem has an infinite run $(q_0, \bar{v}_0), (q_1, \bar{v}_1), \ldots$ where $\bar{0} \leq \bar{v}_i \leq \bar{b}$ for all i and some $\bar{b}$ if and only if there are two indices $j < k$ such that $(q_j, \bar{v}_j) = (q_k, \bar{v}_k)$. In other words, there is a cycle such that the accumulated weight on that cycle is exactly $\bar{0}$. A result in [KS88] shows that the existence of such zero-weight cycle is decidable in polynomial time.

Assume without loss of generality that the given weighted automaton contains only states reachable (while disregarding the weights) from the initial state q_0. It is now clear that if the weighted automaton contains a zero-weight cycle then the parameterized ELU problem has a solution by choosing an appropriate initial weight vector $\bar{v}_0$ and a sufficiently large upper bound $\bar{b}$ which enables us to execute the whole cycle plus reach the cycle from the initial pair $(q_0, \bar{v}_0)$. On the other hand, if there is no zero-weight cycle then the parameterized ELU does not have a solution, as for any choice of $\bar{v}_0$ and $\bar{b}$, every run will eventually violate either the lower bound or the upper bound.

By similar arguments, it is easy to see that a parameterized ELW problem has a solution if and only if the weighted automaton contains a nonnegative-weight cycle. To check for the existence of such a cycle in polynomial time we can use the trick described in [CDHR10]. We simply add to each state in the automaton a number of self-loops with weights $(-1, 0, \ldots, 0)$, $(0, -1, 0, \ldots, 0)$, $\ldots (0, \ldots, 0, -1)$ and then ask for the existence of a zero-weight cycle. ■

However, if the upper bound can appear as a weight, we get undecidability.

Recall that a *Minsky machine* with two nonnegative counters c_1 and c_2 is a sequence of labeled instructions $1 : \mathsf{inst}_1;\ 2 : \mathsf{inst}_2;\ \ldots, n : \mathsf{inst}_n$ where $\mathsf{inst}_n = \mathsf{HALT}$ and each inst_i, $1 \leq i < n$, is of one of the following forms:

(Inc) $\quad i\texttt{: } c_j \texttt{ := } c_j \texttt{ + 1; goto } k$

(Test-Dec) $\quad i\texttt{: if } c_j \texttt{ = 0 then goto } k \texttt{ else } (c_j \texttt{ := } c_j \texttt{ - 1; goto } \ell)$

for $j \in \{1, 2\}$ and $1 \leq k, \ell \leq n$. Instructions of type (Inc) are called *increment* instructions and of type (Test-Dec) are called *test and decrement* instructions. A configuration is a triple (i, v_1, v_2) where i is the current instruction and v_1 and v_2 are the values of the counters c_1 and c_2 respectively. A computation step between configurations is defined in the natural way. If starting from the initial configuration $(1, 0, 0)$ the machine reaches the instruction HALT then we say it *halts*.

It is well known that the problem whether a given Minsky machine halts is undecidable [Min67].

Theorem 9. *The parameterized ELU(2) and ELW(4) problems where the upper bound parameter can appear as a weight in the underlying weighted automaton are undecidable.*

Proof. We provide a reduction from the undecidable halting problem of Minsky machines [Min67] to ELU(3). Let $1 : \mathsf{inst}_1;\ 2 : \mathsf{inst}_2;\ \ldots, n : \mathsf{inst}_n$ be a Minsky machine over the nonnegative counters c_1 and c_2. We construct a 3-weighted automaton $(Q, q_0, \longrightarrow)$ where $Q = \{q_i, q_i' \mid 0 \leq i \leq n\}$ and where the initial weight vector $\bar{v}_0$ and the upper bound $\bar{b}$ are parameterized. The intuition is that the first and second coordinates will record the accumulated values of counters c_1 and c_2, respectively, and the third coordinate will be used for counting the number of steps the machine performs. The transitions are of four types:

1. $q_0 \xrightarrow{+\bar{b}} q_0' \xrightarrow{-\bar{b}} q_1$
2. For each instruction $i\texttt{: } c_j \texttt{ := } c_j \texttt{ + 1; goto } k$, we add the transitions
   - $q_i \xrightarrow{(+1,0,+1)} q_k$ if $j = 1$, and $q_i \xrightarrow{(0,+1,+1)} q_k$ if $j = 2$.
3. For each instruction $i\texttt{: if } c_j \texttt{ = 0 then goto } k \texttt{ else } (c_j \texttt{ := } c_j \texttt{ - 1; goto } \ell)$, we add the transitions
   - $q_i \xrightarrow{(+\bar{b}[1],0,0)} q_i' \xrightarrow{(-\bar{b}[1],0,+1)} q_k$ and $q_i \xrightarrow{(-1,0,+1)} q_\ell$ if $j = 1$, and

- $q_i \xrightarrow{(0,+\bar{b}[2],0)} q_i' \xrightarrow{(0,-\bar{b}[2],+1)} q_k$ and $q_i \xrightarrow{(0,-1,+1)} q_\ell$ if $j = 2$.

4. Finally, we add the loop $q_n \xrightarrow{(0,0,0)} q_n$.

It is now easy to argue that the constructed 3-weighted automaton has an infinite run if and only if the Minsky machine halts.

From Theorem 5 we get that ELU(3) is reducible to ELU(2), hence the parameterized existential problem is undecidable for vectors of dimension two. By Theorem 3 we can reduce ELU(2) to ELW(4), which implies the undecidability of the problem also for weak upper bound and weight vectors of size at least four. ■

The parameterized problems ELU(1) and ELW(k) for $1 \leq k \leq 3$ where the upper bound parameter can appear in the automata are open.

7. Extension to Timed Automata

It is natural to ask for extensions of the results presented in this article to multiweighted *timed* automata and games [ATP04, BFH+01].

Formally a k-weigthed timed automaton is defined as follows. Let $\Phi(C)$ be the standard set of (diagonal-free) *clock constraints* over a finite set of *clocks* C given by conjunctions of constraints of the form $x \bowtie c$ with $x \in C$, $c \in \mathbb{Z}$, and $\bowtie$ any of the relations $\leq$, $<$, $=$, $>$, and $\geq$.

A k-weighted timed automaton is a tuple $T = (L, \ell_0, C, E, r, w)$, where L is a finite set of locations, $\ell_0 \in L$ is the initial location, C is a finite set of clocks, $E \subseteq L \times \Phi(C) \times 2^C \times L$ is a finite set of edges, and $r : L \to \mathbb{Z}^k$, $w : E \to \mathbb{Z}^k$ assign weight vectors to locations and edges.

Note that we allow weight updates on edges here; as shown in [BFLM10], this can have a significant influence on the complexity of the problems one wants to consider.

We also use the standard notation $v \models g$ for the fact that a *valuation* $v : C \to \mathbb{R}_{\geq 0}$ satisfies the clock constraint $g \in \Phi(C)$, $v{+}t$ for the valuation given by $(v{+}t)(x) = v(x){+}t$, and $v[R]$ for the valuation with clocks in R reset to value 0.

The semantics of a k-weighted timed automaton is now given by a k-weighted automaton with states $Q = L \times \mathbb{R}_{\geq 0}^C$ and transitions

$$(\ell, v) \xrightarrow{t \cdot r(\ell)} (\ell, v + t) \text{ for all } t \in \mathbb{R}_{\geq 0} \text{ (delay)},$$

$$(\ell, v) \xrightarrow{w(e)} (\ell', v') \text{ for all } e = (\ell, g, R, \ell') \in E \text{ s.t. } v \models g \text{ and } v' = v[R] \text{ (switch)}.$$

We recall the fact that weights on delay transitions may be non-integer real numbers; formally we have to change the definition of a k-weighted game to allow an infinite weighted transition relation $\longrightarrow \subseteq Q \times \mathbb{R}^k \times Q$. A *run* in a multiweighted timed automaton is a sequence of alternating switch and delay transitions in the corresponding multiweighted automaton.

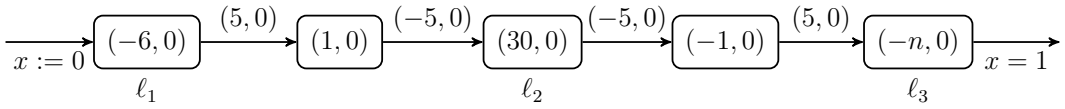


Figure 8.: The module for incrementing ($n = 3$) and decrementing ($n = 12$)

For the case with one weight and one clock only, extensions to timed automata have been discussed in [BFLM10, BFL+08]. In [BFL+08] it has been shown that the GLU(1) problem is already undecidable for one-clock multiweighted timed automata. By an adaptation of the technique introduced in [BFL+08], we can prove that the *existential* problem ELU with two weights and one clock is also undecidable. As the reductions from Theorem 3 apply also to timed automata, we altogether get the following undecidability results.

Theorem 10. *The problems ELU(2), EL(4) and ELW(4), and GLU(1), GL(2) and GLW(2) are undecidable for one-clock multiweighted timed automata.*

Proof of Theorem 10. We start by proving the case of ELU(2). The proof is by reduction from Minsky machines to multiweighted timed automata, based on the technique of the proof of Theorem 17 in [BFL+08]. We construct a one-clock 2-multiweighted timed automaton T that simulates a Minsky machine such that the Minsky machine loops if and only if T is a positive instance of the ELU(2) problem.

The values c_1, c_2 of the counters will be encoded by the accumulated weight vector $\bar{w} = (5 - 2^{-c_1}, 5 - 2^{-c_2})$ and T will start with an initial weight vector of $\bar{v}_0 = (4, 4)$, and the upper bound vector is $\bar{b} = (5, 5)$.

In order to simulate the instructions of the Minsky machine we now describe two different modules of T.

Increment and decrement: Figure 8 shows the general module used for incrementing and decrementing counter c_1; by interchanging the two weights one obtains the module for c_2. Note that the second component $\bar{w}[2]$ of the weight vector is not changed in the module, and we assume that $\bar{w}[1] = 5 - e$ when entering the module and $0 \leq en \leq 30$. We now prove that when exiting the module, $\bar{w}[1] = 5 - \frac{en}{6}$.

Any legal run must decrease $\bar{w}[1]$ to value 0 while delaying in ℓ_1 (otherwise adding 5 to $\bar{w}[1]$ in the following transition exceeds the upper bound), hence the clock x has the value $\frac{5-e}{6}$ when leaving ℓ_1. We cannot delay in the next location, as this would exceed the upper bound, hence we arrive in ℓ_2 with $x = \frac{5-e}{6}$ and $\bar{w}[1] = 0$. We must delay in ℓ_2 until $\bar{w}[1]$ has the value 5, otherwise the following transition would exceed the lower bound, hence the delay in ℓ_2 is precisely $1/6$ time units. Location ℓ_3 is thus entered with $x = 1 - \frac{e}{6}$ and $\bar{w}[1] = 5$, and after delaying for $e/6$ time units, $\bar{w}[1] = 5 - \frac{en}{6}$.

Hence instantiating $n = 3$ converts an input of $\bar{w}[1] = 5 - e$ to $\bar{w}[1] = 5 - \frac{e}{2}$, thus incrementing counter c_1. Likewise, for $n = 12$ counter c_1 is decremented.

The test-decrement module: We have shown how to implement a module which increments a counter, so we miss to construct a module performing the instruction

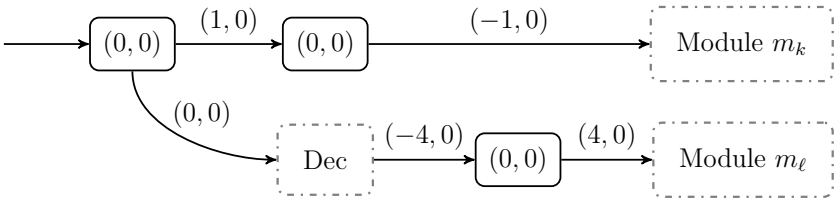


Figure 9.: The test-decrement module

`if` $c_1 = 0$ `then goto` k `else (`c_1 `:=` c_1 `- 1; goto` ℓ`)`. This module is displayed in Figure 9; the construction for the corresponding c_2 module is symmetric.

We now argue that the module acts as claimed. If $c_1 = 0$ when entering, i.e. $\bar{w}[1] = 4$, then the upper path can be taken, leading to Module m_k with counter value $c_1 = 0$ (and c_2 unchanged). On the other hand, attempting to take the lower path exits the Dec module with a value $\bar{w}[1] = 3$, hence the following transition leads to a violation of the lower bound.

If $c_1 \geq 1$, i.e. $\bar{w}[1] \geq 4.5$, when entering the module, then the $(1, 0)$ transition in the upper path will violate the upper bound. In the lower path, the Dec module is left with $\bar{w}[1] \geq 4$ and c_1 decreased by one, hence Module m_ℓ is entered with the correct c_1 value.

We have shown how to faithfully simulate a Minsky machine by a one-clock 2-multiweighted timed automaton such that the Minsky machine has an infinite computation if and only if the timed automaton has an infinite alternating run.

By undecidability of the halting problem for Minsky machines, this concludes the proof for the case of ELU(2).

For the case of EL(4) and ELW(4) we observe that the construction in the proof of Theorem 3 can be adapted also to multiweighted timed automata. Given a k-weighted timed automaton $T = (L, \ell_0, C, I, E, r, w)$ and an upper bound vector $\bar{b}$, we construct a $2k$-weighted timed automaton $T' = (L', \ell_0', C, I', E', r', w')$ with $L' = L \uplus \{\ell_0'\}$, $I'(\ell) = I(\ell)$ for $\ell \in L$, $I'(\ell_0') = (\bigwedge_{x \in C} x = 0)$, $E' = E \cup \{(\ell_0', (\bigwedge_{x \in C} x = 0), \emptyset, \ell_0)\}$, $r'(\ell_0') = \bar{0}$, and

$$
\begin{aligned}
&r'(\ell) = (\bar{r}[1], \ldots, \bar{r}[k], -\bar{r}[1], \ldots, -\bar{r}[k]) \text{ for } \ell \in L \text{ and } \bar{r} = r(\ell),\\
&w'(\ell, g, R, \ell') = (\bar{w}[1], \ldots, \bar{w}[k], -\bar{w}[1], \ldots, -\bar{w}[k])\\
&\qquad\qquad\qquad\qquad \text{for } (\ell, g, R, \ell') \in E \text{ and } \bar{w} = w(\ell, g, R, \ell'),\\
&w'(\ell_0', g, R, \ell_0) = (0, \ldots, 0, \bar{b}[1], \ldots, \bar{b}[k]).
\end{aligned}
$$

Then T is a positive instance of the ELU problem with an upper bound vector $\bar{b}$ if and only if T' is a positive instance of the EL or ELW (with weak upper bound vector $\bar{b}$) problems. The claim then follows from Theorem 10.

The results for the game versions of the problems follow from undecidability of GLU(1) [BFL+08] together with Theorem 3. ■

8. Conclusion and Future Work

We have presented an extension of different types of energy games to a setting with multiple weights and established a comprehensive account of the complexity of these problems. To derive our results, we have demonstrated a close connection of these problems with infinite run problems in Petri nets, together with a number of reductions between different variants of multiweighted energy games. We have also studied a parameterized version of these problems and shown that depending on the precise statement of the problem, it is either solvable in polynomial time or undecidable. Finally, we have demonstrated that for the timed automata extension of energy games, the lower and upper bound existential problem is undecidable already for one clock and two weights.

There are two main problems left open. The first one deals with settling the complexity of the one-weight lower and upper bound existential problem, which is only known to be between NP and PSPACE. This is closely related to the lower bound and weak upper bound problems with a fixed number of weights. The second problem deals with the complexity of energy games with lower bound only, as the present upper complexity bound depends on the number of weights and does not have a matching lower bound. Further extensions with e.g. different acceptance conditions and the optimization problems are also of future interest.

B

Optimal Bounds for Multiweighted and Parametrised Energy Games

Line Juhl Kim G. Larsen
Aalborg University, Department of Computer Science, Denmark

Jean-François Raskin
Université Libre de Bruxelles, Belgium

Abstract Multiweighted energy games are two-player multiweighted games that concern the existence of infinite runs subject to a vector of lower and upper bounds on the accumulated weights along the run. We assume an unknown upper bound and calculate the set of vectors of upper bounds that allow an infinite run to exist. For both a strict and a weak upper bound we show how to construct this set by employing results from previous works, including an algorithm given by Valk and Jantzen for finding the set of minimal elements of an upward closed set. Additionally, we consider energy games where the weight of some transitions is unknown, and show how to find the set of suitable weights using the same algorithm.

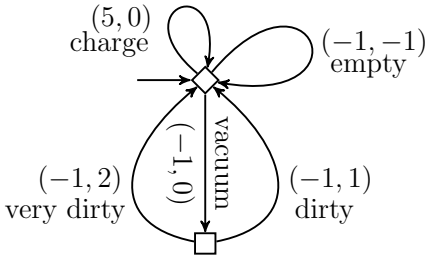
1. Introduction

Energy games have recently attracted considerable attention [BFLM10, BFL+08, BJK10, CdAHS03, Cha10, CD10, CDHR10, DDG+10, FJLS11]. An energy game is played by two players on a weighted game automaton. Player 1 is winning if he has a strategy such that all infinite runs respecting this strategy has nonnegative accumulated weight at all times. A variant of energy games furthermore requires an upper bound that the accumulated weight must stay below at all times in order for Player 1 to win. The upper bound can also be weak, implying that all accumulated weights going above are simply truncated. As embedded systems are often resource-constrained systems exhibiting a reactive behaviour, energy games are relevant for ensuring that the resource of the system never becomes unavailable no matter the choices of the environment. Multiweighted energy games, where the weights of the automaton are vectors, are useful for modelling systems that depend on more than one resource.

In this paper we consider multiweighted energy games with unknown upper bound (both strict and weak) and fixed initial value. When considering the existence of a vector of upper bounds such that Player 1 is winning, it is from an engineering viewpoint relevant to construct the actual vector instead of giving a boolean answer to the problem. We therefore seek to construct the exact set of upper bounds that make Player 1 win the energy game. We will denote such upper bounds as winning. For both types of upper bounds it is clear that if some vector of upper bounds is winning, then also coordinate-wise larger vectors are winning. In order to characterise the set of winning upper bounds, it is thus sufficient only to find the smallest vector of winning upper bounds. However, $\leq$ is not a total order on $\mathbb{Z}^k$ for $k > 1$, so instead of a unique smallest vector we search for the set of smallest incomparable vectors winning for Player 1.

To motivate the study, let us consider a small example of an automatic vacuum cleaner. The machine has a rechargeable battery and a container for the dust it collects. As we are interested in a behaviour that never empties the battery nor completely fills the dust container, it can be modelled using a 2-weighted energy game as seen in Figure 1a. The vector attached to each transition denotes the change in battery (first coordinate) and container level (second coordinate). The diamond state is controlled by Player 1 while the square state is controlled by the environment, as the vacuum cleaner does not control how dirty the floor is when vacuuming.

The lower bound of the two resources are naturally 0, while the upper bound corresponds to the size of the battery and container, respectively. For a manufacturer it is useful to know what size he can possible make the battery and the container in order to ensure an infinite run. The set of minimal winning upper bounds consists in this case of the vectors $(6, 2)$ and $(5, 3)$. The upper bound vector $(6, 2)$ keeps the container as small as possible, while the upper bound vector $(5, 3)$ keeps the battery as small as possible. Surely, the first coordinate of an upper bound cannot be smaller than 5, as charging adds 5 to the accumulated weight in the first coordinate. Similarly, the second coordinate cannot be smaller than 2, as a very dirty floor adds 2 to the accumulated weight in the second coordinate. The winning strategy for Player 1 can be seen in Figure 1b for $(6, 2)$ and Figure 1c for $(5, 3)$. Any vector larger than one of the minimal vectors will



(a) A vacuum cleaner 2-weighted game

if *battery* ≥ 1 and *container* ≥ 1
then empty
else if *battery* ≥ 2 and *container* $= 0$
then vacuum
else charge

(b) A winning strategy for Player 1 with upper bound $(6, 2)$

if *battery* $= 0$
then charge
else if *battery* ≥ 2 and *container* ≤ 1
then vacuum
else empty

(c) A winning strategy for Player 1 with upper bound $(5, 3)$

Figure 1.: A vacuum cleaner example

also serve as a winning upper bound.

Contributions. For multiweighted energy games with an unknown upper bound (both strict and weak) and fixed initial value we calculate the set of minimal upper bounds such that the energy game is winning. For a strict upper bound we make use of results from [BJK10] and [FJLS11] in order to construct the set, yielding an algorithm running in $2k$-exponential time. In the case of a weak upper bound we utilise an algorithm given by Valk and Jantzen in [VJ84], that constructs the set of minimal elements of an upward-closed set (the so-called Pareto frontier), by showing that the preconditions for applying the algorithm are fulfilled. The relevant definitions are given in Section 2, while Section 3 and Section 4 treat the cases of a weak and a strict upper bound, respectively.

Furthermore we study a related problem in Section 5, where we consider multiweighted energy games where both the initial value and the upper bound (if any) are known, but where some weights of the transitions are unknown. We call these parametrised transitions. We here seek to characterise the set of possible evaluations for the parameters such that Player 1 can win the energy game. For a weak upper bound, it is again sufficient to construct the set of minimal evaluations such that Player 1 is winning, and we are once again able to apply the algorithm from [VJ84] to construct the set.

Related Work. Previously energy games have been considered in different settings. One-weighted energy games with both upper and lower bounds were defined in [BFL+08]. Here they study the existence of a winning strategy for Player 1 for a fixed initial value and fixed upper and lower bound and provide bounds on the complexity for the identified problems both in a timed and untimed setting. The paper [FJLS11] extends the results

from [BFL+08] to the multiweighted case. The work of [CDHR10] treats multiweighted energy games with only a lower bound and show that deciding whether there exists a vector of initial values for the resources such that Player 1 can win the energy game is coNP-complete and that only finite-memory strategies are sufficient. In [BJK10] they give a procedure running in $(k-1)$-exponential time that calculates the Pareto frontier of winning initial vectors in multiweighted energy games with k weights, a lower bound and unary weights on transitions (vector addition systems with states). For energy games with imperfect information and fixed initial value, the paper [DDG+10] proves decidability of the problem, but undecidability in case the initial value is not fixed.

2. Multiweighted Energy Games

In this paper, we let $\mathbb{Z}$ and $\mathbb{N}$ denote the sets of all integers and all nonnegative integers, respectively. We define $\mathbb{N}_\omega$ as $\mathbb{N} \cup \{\omega\}$, where ω is a new element modelling an arbitrary nonnegative integer. Thus $\omega > m$ for any $m \in \mathbb{N}$.

For two k-dimensional vectors $\bar{v}, \bar{v}' \in \mathbb{N}_\omega^k$ we use the notation $\bar{v}[i]$ to denote the ith coordinate of the vector $\bar{v}$ $(1 \leq i \leq k)$ and write $\bar{v} \leq \bar{v}'$ if $\bar{v}[i] \leq \bar{v}'[i]$ for all $i \in \{1, \ldots, k\}$. We define the sum of two vectors as the coordinate-wise sum, i.e. $\bar{v} + \bar{v}' = (\bar{v}[1] + \bar{v}'[1], \ldots, \bar{v}[k] + \bar{v}'[k])$. The notation $\bar{0} = (0, \ldots, 0)$ is used to denote the vector of all zeros and $\overline{\infty} = (\infty, \ldots, \infty)$ as the ditto for ∞.

A set $K \subseteq \mathbb{N}^k$ is said to be *upward closed* if $\bar{x} \in K$ and $\bar{x} \leq \bar{y}$ implies $\bar{y} \in K$. Furthermore we define $min(K)$ as the set of smallest incomparable vectors of K,

$$min(K) = \{\bar{x} \in K \mid \forall\, \bar{y}(\neq \bar{x}) \in K : \bar{y} \not\leq \bar{x}\}\ .$$

We call such a $min(K)$ the *minimal generating set* of K.

It it well-known that such a set of incomparable vectors of natural numbers will be finite (as stated in Dickson's lemma) and unique.

We now define a game with multiple weights as an automaton with dedicated state sets for each player and a transition function decorated with a vector of integers.

Definition 1. A k-weighted game is a four-tuple $G = (S_1, S_2, s_0, \longrightarrow)$, where S_1 and S_2 are finite, disjoint sets of existential and universal states, respectively, $s_0 \in S_1 \cup S_2$ is the start state and $\longrightarrow \subseteq (S_1 \cup S_2) \times \mathbb{Z}^k \times (S_1 \cup S_2)$ is a finite multiweighted transition relation.

We write $s \xrightarrow{\bar{w}} s'$ whenever $(s, \bar{w}, s') \in \longrightarrow$. In the following we consider only non-blocking automata, i.e. for every $s \in S_1 \cup S_2$ we have $s \xrightarrow{\bar{w}} s'$ for some $\bar{w} \in \mathbb{N}^k$ and $s' \in S_1 \cup S_2$.

Definition 2. A configuration in a k-weighted game $G = (S_1, S_2, s_0, \longrightarrow)$ is a pair $(s, \bar{v})$ such that $s \in S_1 \cup S_2$ and $\bar{v} \in \mathbb{Z}^k$.

A weighted run π in G restricted to a weak upper bound $\bar{b} \in (\mathbb{N} \cup \{\infty\})^k$ is an infinite sequence of configurations $(s_0, \bar{v}_0), (s_1, \bar{v}_1), \ldots$ such that for all $i \geq 0$ we have $s_i \xrightarrow{\bar{w}_i} s_{i+1}$ and $\bar{v}_{i+1}[j] = \min\{\bar{b}[j], \bar{v}_i[j] + \bar{w}_i[j]\}$ for all $j \in \{1, \ldots, k\}$.

By $\mathsf{WR}_{\bar{b}}(G)$ we denote all weighted runs in G with weak upper bound $\bar{b}$ starting from the initial state. Let π_i denote the ith configuration of a weighted run π.

As we are concerned with games we need a notion of a strategy for a player.

Definition 3. A strategy for Player $i \in \{1, 2\}$ in a k-weighted game $G = (S_1, S_2, s_0, \longrightarrow)$ restricted to some weak upper bound $\bar{b}$ is a mapping σ assigning a configuration $(s, \bar{v})$ to any finite prefix of a weighted run in $\mathsf{WR}_{\bar{b}}(G)$ of the form $(s_0, \bar{v}_0), \ldots, (s_j, \bar{v}_j)$ where $s_j \in S_i$ such that $(s_0, \bar{v}_0), \ldots, (s_j, \bar{v}_j), (s, \bar{v})$ is a prefix of a weighted run in $\mathsf{WR}_{\bar{b}}(G)$.

We say that a weighted run $(s_0, \bar{v}_0), (s_1, \bar{v}_1), \ldots$ respects a strategy σ of Player i if $\sigma((s_0, \bar{v}_0), \ldots, (s_j, \bar{v}_j)) = (s_{j+1}, \bar{v}_{j+1})$ for all $s_j \in S_i$.

We can now define the following three notions of winning vectors.

GL: Given a k-weighted game G, a vector $\bar{v}_0 \in \mathbb{N}^k$ *wins* the (multiweighted) energy game with lower bound (GL) if there exists a winning strategy σ for Player 1 such that any weighted run $(s_0, \bar{v}_0), (s_1, \bar{v}_1), \ldots \in \mathsf{WR}_{\overline{\infty}}(G)$ respecting σ satisfies $\bar{0} \leq \bar{v}_i$ for all $i \geq 0$.

GLW: Given a k-weighted game G, a vector $\bar{b} \in \mathbb{N}^k$ *wins* the (multiweighted) energy game with lower and weak upper bound (GLW) if there exists a winning strategy σ for Player 1 such that any weighted run $(s_0, \bar{0}), (s_1, \bar{v}_1), \ldots \in \mathsf{WR}_{\bar{b}}(G)$ respecting σ satisfies $\bar{0} \leq \bar{v}_i$ for all $i \geq 0$.

GLU: Given a k-weighted game G, a vector $\bar{b} \in \mathbb{N}^k$ *wins* the (multiweighted) energy game with lower and upper bound (GLU) if there exists a winning strategy σ for Player 1 such that any weighted run $(s_0, \bar{0}), (s_1, \bar{v}_1), \ldots \in \mathsf{WR}_{\overline{\infty}}(G)$ respecting σ satisfies $\bar{0} \leq \bar{v}_i \leq \bar{b}$ for all $i \geq 0$.

Notice that we may allow an initial weight vector $\bar{v}_0$ different from $\bar{0}$. This is evident by adding a new start state with one transition labelled with $\bar{v}_0$ pointing to the old start state.

Define $I = \{\bar{v}_0 \in \mathbb{N}^k \mid \bar{v}_0 \text{ wins GL}\}$, $W = \{\bar{b} \in \mathbb{N}^k \mid \bar{b} \text{ wins GLW}\}$ and $U = \{\bar{b} \in \mathbb{N}^k \mid \bar{b} \text{ wins GLU}\}$ as the winning vectors for GL, GLW and GLU, respectively. The paper [BJK10] calculates the set $min(I)$ using $(k-1)$-exponential time for k-weighted energy games with unary weights.

This paper aims to calculate the minimal generating sets of winning weak and strict upper bounds, $min(W)$ and $min(U)$.

3. Weak Upper Bound

In this section we study the problem of finding the set $min(W)$ as defined in Section 2.

The paper [VJ84] by Valk and Jantzen contains an algorithm for computing the minimal generating set of an upward closed set $K \subseteq \mathbb{N}^k$ provided that K satisfies a certain decidability criterion.

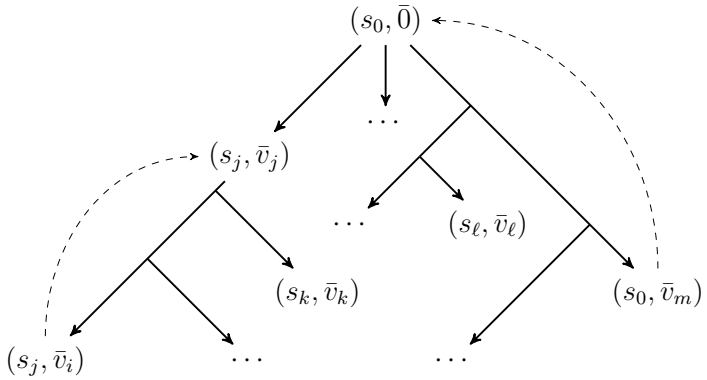


Figure 2.: Self-covering tree

The decidability question is defined for a set $K \subseteq \mathbb{N}^k$ as a predicate $p_K : \mathbb{N}_\omega^k \to \{\texttt{true}, \texttt{false}\}$ by $p_K(\bar{d}) = (\{\bar{d}' \in \mathbb{N}^k \mid \bar{d}' \leq \bar{d}\} \cap K \neq \emptyset)$. Thus $p_K(\bar{d})$ decides whether or not the set K has any elements in common with the set of vectors smaller than or equal to $\bar{d}$. If $p_K(\bar{d})$ is decidable for any $\bar{d} \in \mathbb{N}_\omega^k$, the algorithm can be applied to compute the minimal generating of K.

We will now argue that the algorithm proposed in [VJ84] is useful for constructing $min(W)$. The set W is upward closed since a weak upper bound $\bar{b} \in \mathbb{N}^k$ that wins GLW ensures that any $\bar{b}' \geq \bar{b}$ will also win GLW. As $min(W)$ is exactly the minimal generating set of W, $min(W)$ can be found using the algorithm in case p_W is decidable.

Lemma 4. *The predicate $p_W(\bar{d})$ is decidable for any $\bar{d} \in \mathbb{N}_\omega^k$.*

Proof. Given a vector $\bar{d} \in \mathbb{N}_\omega^k$ the following procedure will either construct $\bar{b} \in W$ such that $\bar{b} \leq \bar{d}$ or report than no such $\bar{b}$ exists. Let $\bar{d}'$ be the vector $\bar{d}$ where all ω-entries are substituted by ∞.

Starting from the configuration $(s_0, \bar{0})$ we construct a self-covering tree containing prefixes of all weighted runs. Any configuration $(s, \bar{v})$ induces the child $(s', \bar{v}')$ for any $s \xrightarrow{\bar{w}} s'$ such that $\bar{v}'[\ell] = \min\{\bar{d}'[\ell], \bar{w}[\ell] + \bar{v}[\ell]\}$ for all $\ell \in \{1, \ldots, k\}$. The unfolding of the game graph stops for each branch (i.e. weighted run $(s_0, \bar{0}), (s_1, \bar{v}_1), \ldots \in \mathsf{WR}_{\bar{d}'}(G)$) when reaching an i such that either

A. $\bar{v}_i[\ell] < 0$ for some $\ell \in \{1, \ldots, k\}$ or

B. $s_i = s_j$ and $\bar{v}_i \geq \bar{v}_j$ for some $j < i$.

Figure 2 illustrates such a self-covering tree. Here $\bar{v}_i \geq \bar{v}_j$ and $\bar{v}_m \geq \bar{0}$. Any leaf satisfies either A or B.

Notice that since the state set is finite and $(\mathbb{N}^k, \leq)$ is a wqo, such an i exists for all branches and we thus construct a finite tree. In the case of A we mark a leaf configuration $(s_i, \bar{v}_i)$ as losing and in case of B we mark $(s_i, \bar{v}_i)$ as winning. We propagate the marking of the leaves to the configuration $(s_0, \bar{0})$ in the following way, starting with configurations having only leaves as children. If the state of the configuration belongs to Player 1 and

at least one child is winning, we mark the configuration as winning. Otherwise it is losing. If the state of the configuration belongs to Player 2 and all children are winning we mark the configuration as winning. Otherwise it is losing. In case $(s_0, \bar{0})$ is losing, $p_W(\bar{d}) = \texttt{false}$, as any weighted run is forced to a losing leaf if Player 2 consistently picks losing children. If $(s_0, \bar{0})$ is winning, we set $p_W(\bar{d}) = \texttt{true}$, as we can construct $\bar{b}$ and a winning strategy σ for Player 1, proving the existence of a winning vector $\bar{b}$ for GLW.

The strategy σ is determined by the tree. For each prefix of each branch $\pi_{\downarrow n} = (s_0, \bar{0}), \ldots, (s_n, \bar{v}_n)$, where $s_n \in S_1$, we let $\sigma(\pi_{\downarrow n}) = (s, \bar{v})$, where $s_n \xrightarrow{\bar{w}} s$ for some $\bar{w}$ such that $\bar{v} = \bar{v}_n + \bar{w}$ and $(s, \bar{v})$ is a winning child of $(s_n, \bar{v}_n)$. For any branch $\pi_{\downarrow m} = (s_0, \bar{0}), \ldots, (s_n, \bar{v}_n), \ldots, (s_m, \bar{v}_m)$, where $s_m = s_n$ and $\bar{v}_m \geq \bar{v}_n$ (a winning leaf) we let $\sigma(\pi_{\downarrow m}) = \sigma(\pi_{\downarrow n})$. Notice that if $(s_n, \bar{v}_n)$ does not have any winning children (or is a losing leaf) a winning strategy will never lead us to this state (and thus any next state can be picked).

It is easy to see that any weighted run respecting σ keeps all accumulated weights nonnegative, since all transitions taken by Player 1 and 2 leads to states marked as winning by the definition of σ. At some point the weighted run will enter a loop that has a nonnegative accumulated weight in all coordinates. Furthermore σ is finitely representable.

The weak upper bound $\bar{b}$ that satisfies $\bar{b} \leq \bar{d}$ and is contained in W can be found by examining the self-covering tree and pruning the tree by removing the branches not respecting σ. For the entries of $\bar{d}$ that are not ω we reuse these entries in $\bar{b}$ and for any remaining ω-entry in dimension ℓ we find the largest accumulated weight $\max_\ell$ seen in any configuration of the tree in dimension ℓ. Formally

$$\bar{b}[\ell] = \begin{cases} \max_\ell & \text{if } \bar{d}[\ell] = \omega \,, \\ \bar{d}[\ell] & \text{otherwise} \end{cases}$$

for all $\ell \in \{1, \ldots, k\}$. This bound is save to apply as a weak upper bound, since truncating all weights at this upper bound will not cause the accumulated weights to be negative at any point. ■

As Lemma 4 allows us to use the algorithm presented in [VJ84] we get the following corollary.

Corollary 5. *The set $min(W)$ is computable.*

Notice that we can apply the above procedure for the special case of $\bar{d} = (\omega, \ldots, \omega)$ if we are interested in whether there exists some weak upper bound that wins GLW (e.g. determine whether W is empty or not).

4. Strict Upper Bound

For the case of a strict upper bound we see that U is also an upward closed set, but for deciding $p_U(\bar{d})$ for any $\bar{d} \in \mathbb{N}_\omega^k$ we cannot use the approach presented in the proof of

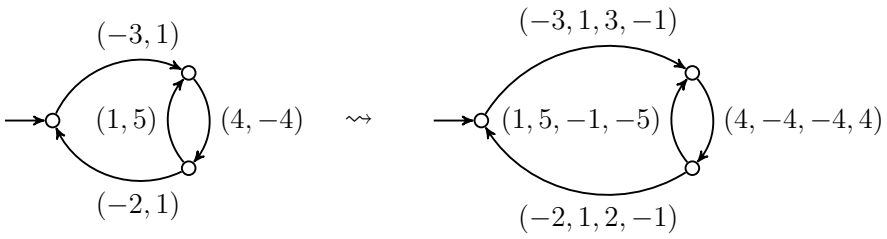


Figure 3.: Example of a reduction from G_k to G_{2k}

Lemma 4. This is due to the construction of the self-covering tree, where we here cannot stop when reaching a cycle with positive accumulated weight in one of the coordinates, since looping forever (as indicated in Figure 2 by dashed arrows) will eventually cause one of the strict upper bounds to be violated. For constructing $min(U)$ we instead make use of energy games with no upper bound.

Theorem 6. *The set $min(U)$ is computable in $2k$-exponential time.*

Proof. The paper [FJLS11] provides the following useful reduction. Determining whether a given upper bound $\bar{b}$ wins GLU with k weights is polynomial time reducible to determining whether the initial vector $(\bar{0}[1], \ldots, \bar{0}[k], \bar{b}[1], \ldots, \bar{b}[k])$ wins GL with $2k$ weights.

Given a k-weighted game G_k the reduction works by constructing a $2k$-weighted game G_{2k} by doubling the number of weights on each transition of G_k, adding a new start state and letting each new transition have the weight $(\bar{w}[1], \ldots, \bar{w}[k], -\bar{w}[1], \ldots, -\bar{w}[k])$ for any old transition with weight $\bar{w}$. The reduction is seen in Figure 3, where the circular states denote either a Player 1 or Player 2 state. Now if one of the first k weights goes above $\bar{b}$, one of the last k weights will go below 0.

The paper [BJK10] provides an algorithm running in $(k-1)$-exponential time for constructing the set $min(I)$ for any k-weighted game with only unary updates, that is a game $G = \{S_1, S_2, s_0, \longrightarrow\}$, where each $s \xrightarrow{\bar{w}} s'$ satisfies $\bar{w} \in \{-1, 0, -1\}^k$.

We can reduce G_{2k} with arbitrary updates to the unary setting by introducing intermediate transitions that repeatedly add or subtract 1 (causing an exponential blowup in the size of the automaton) and obtain the finite set $min(I)$ by applying the algorithm presented in [BJK10].

Now we can easily construct $min(U)$ for G_k from $min(I)$ for G_{2k} as "subvectors" of the vectors in $min(I)$ with all 0's in the first k coordinates,

$$min(U) = \{\bar{b} \in \mathbb{N}^k \mid (\bar{0}[1], \ldots, \bar{0}[k], \bar{b}[1], \ldots, \bar{b}[k]) \in min(I)\} .$$

This procedure presented in [BJK10] runs in $(k-1)$-exponential time for en k-weighted game with unary updates on transitions. Since we in our setting double the number of weights and reduce the arbitrary weights to unary weights, we achieve a procedure running in $2k$-exponential time. ∎

In case of an energy game with both unknown strict upper bound and unknown initial value, the above proof can as well be applied for finding the set of all pairs of initial

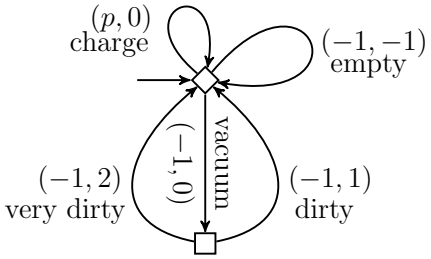


Figure 4.: A parametrised vacuum cleaner example

values and upper bounds that will win the energy game (this set corresponds to the set $min(I)$ for G_{2k}). The problem of simultaneous synthesis with initial value and strict upper bound can therefore be solved in $2k$-exponential time.

5. Parametrised Transitions

A variant of the problem of parametrised bounds is parametrised transitions. Instead of letting the upper bound or initial value be unknown, we may also consider multiweighted energy games where not all weights of the transitions that gain resources are known. As in the case of an unknown upper bound, we are interested in not only knowing whether there exists an assignment of weights such that Player 1 has a winning strategy in the various energy games, but in constructing the actual set of assignments such that Player has a winning strategy. For no upper bound or a weak upper bound this set is upward closed and can thus be characterised by its minimal generating set. For a strict upper bound this is not the case and we must to represent the set otherwise.

Consider the automatic vacuum cleaner in Figure 4, where the first coordinate of the weight of the charge transition is unknown (the parameter p). For a strict upper bound of $(5, 3)$ we now seek to compute the set of possible weight assignments to p such that Player 1 has a winning strategy. The smallest possible weight assigned to p is 2 (using the strategy from Figure 1c). As it turns out, both 3, 4 and 5 as the value of p give rise to a winning strategy for Player 1 (again as seen in Figure 1c). Surely p cannot be assigned a larger weight than the upper bound in the first coordinate, as this would enable us from charging at any time. The full set of suitable values of p is therefore $\{2, 3, 4, 5\}$.

Formally we let a parametrised k-weighted game $G = \{S_1, S_2, s_0, \longrightarrow\}$ over a set of parameters $P = \{p_1, \ldots, p_\ell\}$ be a game where $\longrightarrow \subseteq (S_1 \cup S_2) \times (\mathbb{Z} \cup P)^k \times (S_1 \cup S_2)$. Given an evaluation function $e : P \to \mathbb{N}$, we let $\longrightarrow_e$ be the set $\longrightarrow$ where any parameter $p_i \in P$ is substituted with its evaluation $e(p_i)$. In case there exists an evaluation function e for a parametrised game G with upper bound $\bar{b}$ such that $\bar{b}$ wins GLU given the game $G_e = (S_1, S_2, s_0, \longrightarrow_e)$, we say that e *wins* GLU with parametrised transitions. The same winning notion can be defined for GLW and GL with parametrised transitions. Given two evaluations e, e', we say that $e \leq e'$ if $e(p_i) \leq e'(p_i)$ for all $i \in \{1, \ldots, \ell\}$.

We denote the set of winning evaluations for GLU, GLW and GL with parametrised transitions by U_T, W_T and I_T, respectively. Notice that the sets W_T and I_T are upward closed, implying that these sets can be characterised by their minimal generating set of evaluations, $min(W_T)$ and $min(I_T)$.

For $min(W_T)$ and $min(I_T)$ we as in Section 3 seek to use the algorithm presented in [VJ84] to construct the two sets. The predicates p_{W_T} and p_{I_T} must decide for any parametrised game G whether there exists an evaluation for G in W_T or I_T, respectively.

Lemma 7. *The predicates $p_{W_T}(\bar{d})$ and $p_{I_T}(\bar{d})$ are decidable for any $\bar{d} \in \mathbb{N}_\omega^\ell$.*

Proof. Given a k-weighted game G with parametrised transitions and either a weak upper bound $\bar{b}$ or no upper bound, the existence of a winning evaluation implies the existence of a winning evaluation e that for all $i \in \{1, \ldots, \ell\}$ satisfies $e(p_i) \leq M$, where M is the largest sum obtained by adding all negative weight updates in one coordinate, i.e. $M = \max_{j \in \{1,\ldots,k\}} \left(\sum_{s \xrightarrow{\bar{w}} s'} \max(0, -\bar{w}[j]) \right)$. To see this we note that between each subsequent visit to any state we only need to visit all other states at most once (otherwise we could remove a loop or Player 2 could force an arbitrary low accumulated weight), and thus we subtract at most M in each coordinate between subsequent visits. Setting $e(p_i) = M$ for all $i \in \{1, \ldots, \ell\}$ we can apply the decidability results from [FJLS11] on the game G_e (either with or without $\bar{b}$ as weak upper bound). Thus p_{W_T} and p_{I_T} can be answered. ■

Using the algorithm presented in [VJ84], this leads to the following corollary.

Corollary 8. *The sets $min(W_T)$ and $min(I_T)$ are computable.*

In the case of GLU with parametrised transitions and $\bar{b}$ as the upper bound the set U_T is not upward closed. However, the set of useful evaluations is finite, since any winning evaluation must satisfy $-\bar{b}[i] \leq e(t)[i] \leq \bar{b}[i]$ for all transitions t and all $i \in \{1, \ldots, k\}$. This set $min(U_T)$ can therefore be constructed by an exhaustive search of the possible winning evaluations (again using the decidability results from [FJLS11]).

6. Conclusion and Future Work

Using the algorithm of Valk and Jantzen [VJ84] we have shown how to characterise the set of winning upper bounds for multiweighted energy games with fixed initial value and a weak upper bound. For a strict upper bound the problem is solvable using $2k$-exponential time. Furthermore we have studied multiweighted energy games with parametrised transitions. For a fixed initial value and either a weak upper bound or no upper bound the same algorithm is applied to construct the set of winning evaluations. For a strict upper bound the set is shown computable as well.

Future work should include an investigation of the complexity of the above problems. As there is no upper bound on the complexity of the Valk and Jantzen algorithm, we have so far no complexity results for the results relying on the algorithm. Another future work regards parametrised transitions, where we so far are able to synthesise

only nonnegative weights, and thus require that weights known to be negative are not parametrised. A likely expansion is therefore to synthesising the set of winning evaluations for energy games where any weight coordinate can be unknown. Furthermore the subject of simultaneous synthesis should be explored, where we consider games with combinations of parametrised values, this may be initial value, upper bound (strict or weak) or weight coordinates of transitions. Another direction of research is to study the problems in connection with imperfect information.

Modal Transition Systems with Weight Intervals

Line Juhl Kim G. Larsen Jiří Srba
Aalborg University, Department of Computer Science, Denmark

Abstract We propose *weighted modal transition systems*, an extension to the well-studied specification formalism of modal transition systems that allows to express both required and optional behaviours of their intended implementations. In our extension we decorate each transition with a weight interval that indicates the range of concrete weight values available to the potential implementations. In this way resource constraints can be modelled using the modal approach. We focus on two problems. First, we study the question of existence/finding the largest common refinement for a number of finite deterministic specifications and we show PSPACE-completeness of this problem. By constructing the most general common refinement, we allow for a stepwise and iterative construction of a common implementation. Second, we study a logical characterisation of the formalism and show that a formula in a natural weight extension of the logic CTL is satisfied by a given modal specification if and only if it is satisfied by all its refinements. The weight extension is general enough to express different sorts of properties that we want our weights to satisfy.

1. Introduction

Modal transition systems [LT88a] provide a specification formalism which can express both safety and liveness requirements of their implementations—labelled transition systems. This formalism allows for two kinds of transitions to be present, namely *required* (must) transitions and *allowed* (may) transitions. A rather loose specification can then be transformed into a concrete implementable system by a series of refinements. This idea of stepwise refinement is applicable for example for the development of embedded systems. Recent work on modal transition systems includes applications in several different areas like component-based software development [Rac08, BPR09], interface theory [UC04, RBB+09], modal abstractions and program analysis [GHJ01, HJS01, NNN08], and other areas [FS08, WGC09]. An overview article can be found in [AHL+08]. A similar concept has been studied also in the area of software product lines (see e.g. [GLS08b] and [GLS08a]), however, their notion of refinement is syntactic and different from the semantic refinement relation (based on the concepts of simulation/bisimulation) studied in the theory of modal transition systems.

We present an extension of modal transition systems called *weighted* modal transition systems that decorate each transition with an interval containing a range of weights. The idea of modelling quantitative aspects in transition systems is well studied. For example weighted transition systems (see e.g. the book [DKV09]) are a known extension of standard labelled transition systems. Such systems are particularly useful for modelling resource constraints, which are often seen in embedded systems (e.g. fuel/power consumption, price). Weights therefore seem like a natural addition to modal transition systems, in order to combine the benefits of the 'modal' approach with the modelling of quantities. By allowing both negative and positive weights, we are furthermore able to model systems with both resource gains and losses.

Contrary to weighted transition systems, where transitions and/or states are labelled with specific weights, we decorate transitions with sets of weights. This adheres to the idea of a 'loose' specification, since a specification then determines the range of allowed weights instead of the precise weight. The refinement process will then rule out some of the weights, eventually ending up with an implementation containing the final concrete weight.

To motivate the use of weighted modal transition systems as a model for embedded systems, consider an ATM machine. Two clients might each give a specification (or requirements), detailing their allowed and required use of the machine, along with intervals specifying the acceptable power consumption for each option. This is demonstrated in Figure 1 (the two topmost systems). Here the dotted lines denote allowed behavior (i.e. the behaviour that a client is willing to perform), while the solid lines denote required behavior (i.e. the behaviour a client is insisting on). The interval attached to each transition is the interval where the power consumption (or some other cost) must lie in. As we can see, both clients require that a card is inserted. After the insertion, the clients only allow three actions, namely balance, withdraw and transfer. The balance option is required by the left client, while the right client requires that a withdrawal must be possible in an implementable system satisfying the specification. Even though

the left client only specifies a withdrawal as optional, he/she requires that a PIN must be entered in order to continue. After the PIN is accepted, an amount can be withdrawn any number of times. The right client on the other hand specifies that a PIN is only optional, however, that each amount withdrawn must be preceded with re-entering the PIN.

An important problem is now to determine the existence of an implementation satisfying the needs of both clients and giving the exact power consumption for each option, fitting in the consumption requirements made by the clients. We call such an implementation a *common implementation*. As it can be seen, the option of a transfer is allowed by both clients, but since their power consumption intervals are not overlapping, it is not possible to produce a specific system with a power consumption satisfying both clients. The transfer option is, however, not required by any of the clients, and can thus be ignored in a possible common implementation. Since insertion of the card, balance, withdrawal and PIN entering are required behavior for one or the other of the two specifications, these must be present in the implementable system. Instead of constructing just one common implementation, we aim at constructing the most permissive common refinement, so that this refinement encapsulates all common implementations. Figure 1 shows a most permissive common refinement below the two clients' specifications. After entering the PIN a withdrawal is only allowed once, since the right-hand side specification requires that a new amount specification is preceded by a PIN, while the left-hand side specification does not.

Considering the common refinement in Figure 1, one might be interested in knowing whether it is possible to withdraw some amount consuming between 10 and 20 energy units. Since a withdrawal consumes at least 13 energy units and at most 19 (adding the lower and upper interval bounds along the path), this is indeed possible. However, since the transition labelled 'amount' is only optional, some concrete implementations may leave it out. It is therefore desirable to develop a logical setting that guarantees that if some property is true for a given specification then it is also true for all its implementations.

Our contribution consists of a definition of weighted modal transition systems and an extension of the concepts related to modal transition systems to the weighted setting. This includes modal and thorough refinements and the definition of an implementation. Then we study the largest common refinement problem of finite deterministic specifications. A construction computing the conjunction of a given number of finite deterministic specifications is presented, and we show that a given specification is their common refinement if and only if it refines the constructed largest common refinement. We further show that deciding whether a common refinement exists or not is a PSPACE-complete problem.

Our algorithm for the largest common refinement was inspired by the common implementation construction provided in [BKLS09b]. However, we extend this technique to the weighted scenario and more importantly generalize the construction such that we construct the 'most permissive' common refinement, contrary to [BKLS09b] where only the existence of a common implementation was studied. The maximality of our construction hence allows for a stepwise and iterative construction of a common im-

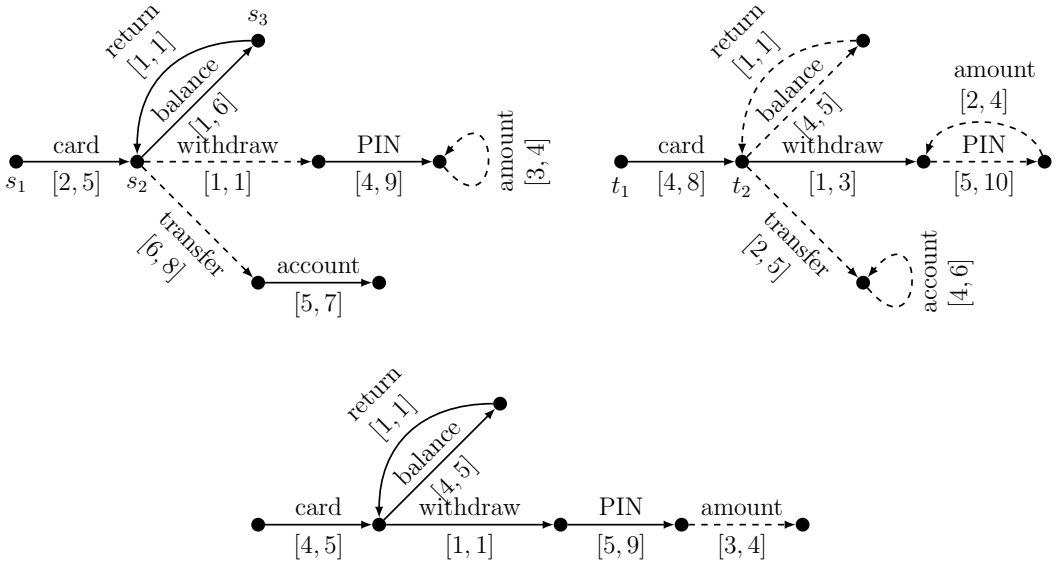


Figure 1.: Two specifications of an ATM machine and their largest common refinement below.

plementation, which is desirable in many applications and was not possible with the previous algorithms.

We note that in this study we restrict ourselves to deterministic specifications as demonstrated, for example, in our running examples. There are two reasons that justify this choice. First of all, for nondeterministic specifications the two studied notions of thorough and modal refinement do not coincide and hence the refinement process, though sound, is not complete (see e.g. [AHL+08]). On the other hand for deterministic specifications, as advocated in the work by Henzinger and Sifakis [HS06, HS07], modal refinement and modal composition are complete. More detailed analysis of this has been recently given in [BKLS09b]. Second, in many practical cases, deterministic specifications are desirable and often used, and much of the recent work deals mainly with deterministic systems. For example in [HS06] the authors discuss two main challenges in embedded systems design: the challenge to build predictable systems, and that to build robust systems. They suggest how predictability can be formalized as a form of determinism, and robustness as a form of continuity.

Another problem we study in this article concerns finding a logical characterisation of weighted modal transition systems. By a natural extension of the action-based CTL we define, based on the work of De Nicola and Vaandrager [NV90], a weighted CTL logic for model checking weighted modal specifications. Compared to other weighted logics like [LLM05] and [DG07], we allow to state arbitrary constraints on the prefixes of model executions and extend the semantics to deal with modal transition systems. On the other hand, we do not consider semiring interpretations of CTL formula quantifiers like in [LLM05] and semiring semantics of MSO like in [DG07].

The definition of our logic is rather generic with respect to the choice for querying the

weight constraints. Our main result shows that a specification satisfies a given formula of weighted CTL if and only if all its refinements satisfy the same formula, which is an important fact that justifies the choice of the logic and supports a step-wise model based development process. We discuss a few specializations of the generic logic to some concrete instances in order to argue for its applicability.

The article is organized as follows. Section 2 introduces the model of weighted modal transition systems, modal and thorough refinement relations and some basic properties of the model. In Section 3 we study the problem of largest common refinement of a given set of finite deterministic specifications and among others prove PSPACE-completeness of the problem. In Section 4 we search for a logical characterisation of weigted modal transition systems. For this purpose we suggest a definition of a generic weighted CTL logic and argue for the soundness of this choice. Finally, Section 5 provides a short summary and mentions some of the open problems.

2. Definitions

We begin by extending the notion of modal transition systems (consult e.g. [LT88a, AHL+08]) by adding an interval to each transition in the specification. This set denotes the different values that the weight of the transition can be instantiated to in an implementation. We define $[n, m] = \{a \in \mathbb{Z} : n \leq a \leq m\}$ for $n \leq m$, $n, m \in \mathbb{Z} \cup \{-\infty, \infty\}$ to denote the closed interval between n and m, and use $\mathcal{I}$ to stand for the set of all such nonempty intervals.

Definition 1. A (interval) weighted modal transition system (WMTS) is a 5-tuple $M = (S, \Sigma, \dashrightarrow, \longrightarrow, \delta)$, where S is a set of states, Σ is an action alphabet, $\longrightarrow \subseteq \dashrightarrow \subseteq S \times \Sigma \times S$ and $\delta : (\dashrightarrow \cup \longrightarrow) \to \mathcal{I}$ assigns a weight interval to transitions. The relations $\dashrightarrow$ and $\longrightarrow$ are called the may and must transitions, respectively.

By the definition of δ we see that if (s, a, t) belongs to both $\dashrightarrow$ and $\longrightarrow$ then the weight intervals of the must and may transition are the same. This fact is important and implicitly used later on. It ensures the so-called *consistency*, meaning that any given modal specification is guaranteed to have an implementation.

We write $s \overset{a}{\dashrightarrow} t$ if $(s, a, t) \in \dashrightarrow$ and $s \overset{a,W}{\dashrightarrow} t$ if $e = (s, a, t) \in \dashrightarrow$ and $\delta(e) = W$, similarly for the elements of $\longrightarrow$. If no t exists such that $(s, a, t) \in \dashrightarrow$, we write $s \not\overset{a}{\dashrightarrow}$, similarly for must transitions. The class of all WMTSs is denoted by $\mathcal{W}$. An WMTS is *deterministic* if for all $s \in S$ and $a \in \Sigma$ there is at most one t such that $(s, a, t) \in \dashrightarrow$. The class of all deterministic WMTSs is denoted by $d\mathcal{W}$.

While a general WMTS models a specification giving a variety of weights and optional behaviour, an implementation (defined below) defines the precise behaviour of the system, including the precise weight of all transitions.

Definition 2 (Implementation)**.** A WMTS is an implementation if $\longrightarrow = \dashrightarrow$ and all weight intervals are singletons. The class of all implementations is denoted by $i\mathcal{W}$.

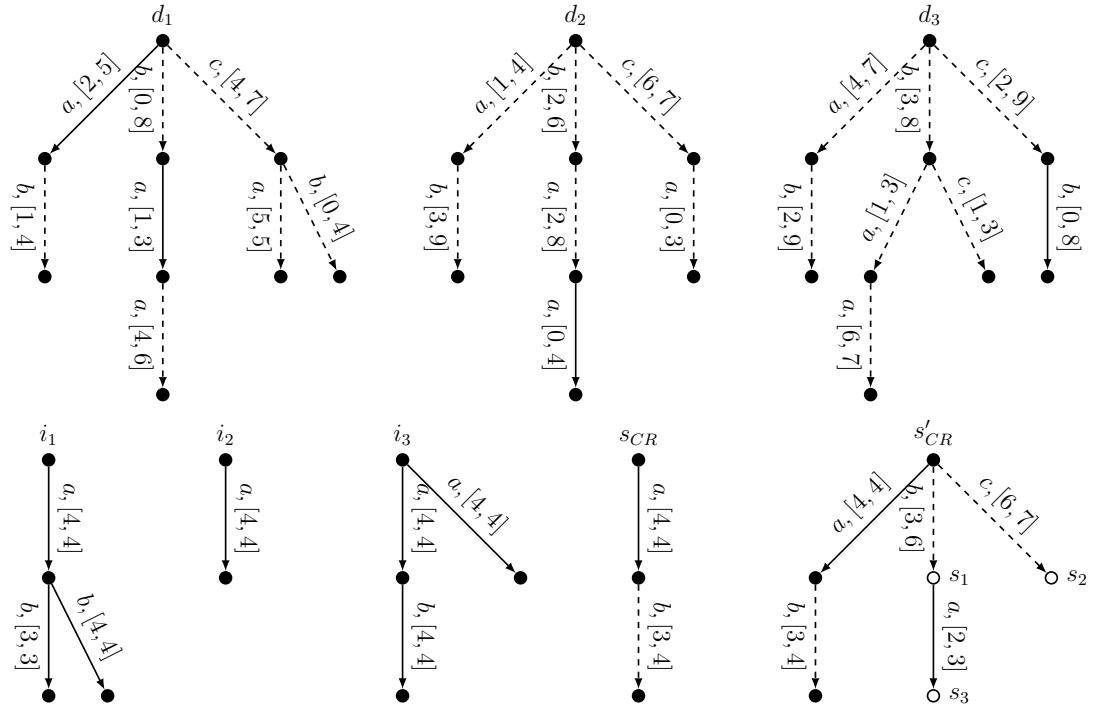


Figure 2.: Different examples of weighted modal transition systems.

To ease the notation, we often denote a WMTS $M = (S, \Sigma, \dashrightarrow, \longrightarrow, \delta)$ containing a state $s \in S$ as a pair, (s, M). Thus the notation $(s, M) \in \mathcal{W}$ is short hand notation for $M \in \mathcal{W}$ with s a state in M (the same applies to $i\mathcal{W}$ and $d\mathcal{W}$). The lowercase letters $s, t, \ldots$ are used for states (specifications) in general, while $i, j, \ldots$ are used for implementations and $d, e \ldots$ are used for deterministic specifications. Since every must transition is also a may transition, may transitions in figures will not be drawn between states if a must transition is already present.

Take a look at the examples in Figure 2 (ignore the systems s_{CR} and s'_{CR} for the moment). The three systems rooted with d_1, d_2 and d_3 are examples of weighted modal transition systems, all of them being deterministic. The systems rooted with i_1, i_2 and i_3 are examples of implementations where may and must transitions coincide and all intervals are singletons.

We can now define the refinement relation for WMTSs, a natural extension of the refinement relation on MTSs. Intuitively, a weight interval on a transition denotes the only acceptable weights allowed in an implementation. A refinement should therefore never allow any new weights to be added, eventually leading to an implementation with only singleton intervals. From now on, when using the term refinement we always refer to the modal refinement relation between two WMTSs as defined below.

Definition 3 (Modal refinement of WMTS)**.** Let $(s_i, M_i) \in \mathcal{W}$ such that $M_i = (S_i, \Sigma, \dashrightarrow_i, \longrightarrow_i, \delta_i)$ for $1 \leq i \leq 2$. We say that s_1 modally refines s_2, written $(s_1, M_1) \leq_m (s_2, M_2)$ or simply $s_1 \leq_m s_2$ if M_1 and M_2 are clear from the context, if there is a refinement relation $R \subseteq S_1 \times S_2$ such that $(s_1, s_2) \in R$ and for each $(s, t) \in R$

and every $a \in \Sigma$:

1. whenever $s \overset{a,W}{\dashrightarrow}_1 s'$, then there exists $t \overset{a,V}{\dashrightarrow}_2 t'$ where $W \subseteq V$ such that $(s', t') \in R$, and
2. whenever $t \overset{a,V}{\longrightarrow}_2 t'$, then there exists $s \overset{a,W}{\longrightarrow}_1 s'$ where $W \subseteq V$ such that $(s', t') \in R$.

Hence (s, M) refines (t, N) ($s \leq_m t$) if it is possible to mimic must transitions in N by M, and it is possible to mimic may transitions in M by N. We say that a WMTS (s, M) is an *implementation of* a WMTS (t, N) if $(s, M) \in i\mathcal{W}$ and $(s, M) \leq_m (t, N)$. Notice that any WMTS has an implementation, for instance one can turn all may transitions into must transitions and pick an arbitrary weight from each interval as the singleton weight.

Consult again Figure 2. The systems i_1, i_2, i_3 and s_{CR} are all refinements of the specification d_1 (in fact also of d_2 and d_3). The first three refinements i_1, i_2 and i_3 are also implementations of d_1.

Remark 1. Notice that for two implementations $(i, I), (j, J) \in i\mathcal{W}$, the relation of modal refinement, $(i, I) \leq_m (j, J)$, corresponds to strong bisimulation (with the assumption that actions and weights are considered as observable pairs).

Definition 4 (Thorough refinement)**.** Let (s, M) be a WMTS and define $[\![s]\!] = \{(i, I) \in i\mathcal{W} : (i, I) \leq_m (s, M)\}$, that is all possible refinements of (s, M) that are also implementations. For $(s, M), (t, N) \in \mathcal{W}$ we say that s thoroughly refines t, written $s \leq_t t$ (or $(s, M) \leq_t (t, N)$), if $[\![s]\!] \subseteq [\![t]\!]$.

The following lemma is easy to prove, but it is an important property that guarantees a sound stepwise refinement development methodology.

Lemma 5. *The relations $\leq_m$ and $\leq_t$ are both transitive.*

Proof. Let $(s, M), (u, O), (t, N) \in \mathcal{W}$. First the case of $\leq_m$. Assume two relations R_1 and R_2 according to Definition 3 showing that $s \leq_m u$ and $u \leq_m t$. It is easy to check that the relation R defined as

$$R = \{(s', t') : \exists u'.((s', u') \in R_1 \wedge (u', t') \in R_2)\}$$

is indeed a refinement relation according to Definition 3 and that $(s, t) \in R$.

We now consider $\leq_t$. Assume $s \leq_t u$ and $u \leq_t t$. This immediately implies that $[\![s]\!] \subseteq [\![u]\!] \subseteq [\![t]\!]$ and hence that $s \leq_t t$. ■

We can now show that modal refinement implies thorough refinement.

Lemma 6. *For two WMTSs, (s, M) and (t, N), it holds that*

$$s \leq_m t \Rightarrow s \leq_t t.$$

Proof. Assume that $s \leq_m t$. If $i \leq_m s$ for an implementation i, then by Lemma 5 also $i \leq_m t$. Hence $[\![s]\!] \subseteq [\![t]\!]$ which means that $s \leq_t t$. ■

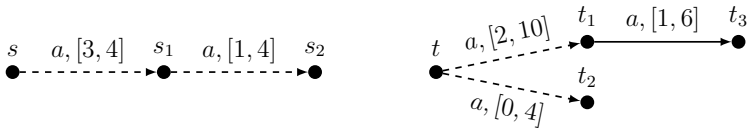


Figure 3.: $s \leq_t t$, but $s \not\leq_m t$.

Notice that thorough refinement does not imply modal refinement. A counter-example can be seen in Figure 3. The figure is overtaken from [BKLS09b] and intervals have been added, thus a counter-example already exists in the unweighted case. To show that $s \not\leq_m t$ we try to construct a relation R. For sure $(s,t) \in R$ must hold. Since $s \overset{a,[3,4]}{\dashrightarrow} s_1$ either (s_1, t_1) or (s_1, t_2) must belong to R. In the first case, $t_1 \overset{a,[1,6]}{\longrightarrow} t_3$ and therefore a must transition from s_1 must exist as well. Since this is not the case, we assume $(s_1, t_2) \in R$. Then the transition $s_1 \overset{a,[1,4]}{\dashrightarrow} s_2$ implies the existence of a may transition from t_2. This is also not the case, thus R cannot exist and $s \not\leq_m t$. On the other hand, every implementation of s can perform at most two consecutive a's with the weights either 3 or 4 for the first a-transition and 1, 2, 3 or 4 for the second a-transition. These implementations are also implementations of t, hence $s \leq_t t$.

However, if we restrict the refined specifications to be deterministic (or at least the right-hand side one) we get the following.

Lemma 7. *For $(s, M) \in \mathcal{W}$ and $(d, D) \in d\mathcal{W}$, it holds that*

$$s \leq_m d \Leftrightarrow s \leq_t d.$$

We omit the proof here, since it follows as a straightforward modification of the proof given in [BKLS09b] by adding appropriate intervals to all transitions.

For the complexity results presented in the remainder of the paper we assume constant time interval operations (the encoding of integers is assumed binary).

In [BKLS09a] it was shown that checking whether a finite modal transition system is thoroughly refined by another finite modal transition system is EXPTIME-complete. The thorough refinement problem for MTSs can be reduced to the same problem for WMTSs by adding the same singleton weight to all transitions. Hence the thorough refinement problem for finite WMTSs is also EXPTIME-hard. The algorithm presented in [BKLS09a] for determining whether one MTS thoroughly refines another one can be easily extended to the weighted setting by adding appropriate checks for set inclusions of the weight intervals. This addition does not effect the running time of the algorithm, and the thorough refinement problem for finite WMTS is therefore decidable in EXPTIME as well.

On the contrary, the problem of deciding whether two finite weighted modal specifications are in the modal refinement relation is decidable in deterministic polynomial time using the standard greatest fixed-point computation, similarly as in the case of strong bisimulation (for efficient algorithms implementing this strategy see e.g. [KS90, PT87]).

3. Largest Common Refinement

This section addresses the *largest common refinement problem* of finite deterministic specifications defined as follows: given a number of finite deterministic WMTSs, $(d_1, D_1), \ldots, (d_n, D_n)$, we want to find a specification $(s, M) \in \mathcal{W}$ such that $(s, M) \leq_m (d_j, D_j)$ for all j, $1 \leq j \leq n$, or to report that no such common refinement exists. Moreover, we are interested in constructing some largest common refinement refinement (s, M) such that any other common refinement of the given deterministic specifications refines (s, M). Notice that such a largest common refinement is not unique. In what follows, we implicitly assume that the given deterministic specifications $(d_1, D_1), \ldots, (d_n, D_n)$ are finite.

Figure 2 shows our running example. Our task is to construct the largest common refinement of the deterministic specifications d_1, d_2 and d_3.

Let $(d_1, D_1), \ldots, (d_n, D_n) \in d\mathcal{W}$ be n deterministic WMTSs. We will construct a specification $(s_{CR}, M_{CR}) \in \mathcal{W}$ and prove that (s_{CR}, M_{CR}) is the most general common refinement of $d_1, \ldots, d_n$. The state set of M_{CR} consists of n-tuples, $(e_1, \ldots, e_n)$, where every e_i belongs the the corresponding state set of D_i. Additionally some states in M_{CR} will be marked. Marked nodes represent situations where no common refinement exists. The pseudo-code in Alg. 1 constructs M_{CR}, a common refinement of the given specifications, or returns that no such refinement exists. The algorithm avoids the construction of the whole product space and returns only the reachable parts of such common refinement.

It is easy to see that the algorithm always terminates. The first repeat-loop (lines 3–22) runs until *Waiting* is empty, and in each iteration one element is removed from *Waiting*. Elements are also added to *Waiting*, however since D_i are finite for all i, and no removed element is added again to *Waiting*, this repeat-loop terminates. The second repeat-loop (lines 23–27) as well as the forall-loop (lines 30 and 31) also terminate due to the finiteness of the set S and finiteness of the $\longrightarrow$ relation.

Alg. 1 constructs M_{CR} by inspecting the n deterministic WMTSs and adding the needed states and transitions to M_{CR}, and furthermore marking states if these represent situations where no common refinement can exist. The marked set is expanded by adding all states, from which a path consisting of only must transitions leads to a marked state. If the state $(d_1, \ldots, d_n)$ is marked, no common refinement exists. If this is not the case, a common refinement exists and the most general common refinement is constructed by removing all marked states and transitions leading to marked states.

In our running example in Figure 2, given the input d_1, d_2 and d_3 Alg. 1 first constructs an intermediate specification s'_{CR} where the marked nodes s_1, s_2 and s_3 are drawn as circles. After removing them the algorithm returns the specification $(s_{CR}, M_{CR},)$.

Lemma 8. *If $(e_1, \ldots, e_n)$, a state in M_{CR}, has a path consisting only of must transitions leading to a state marked by Alg. 1 in the first repeat-loop, then $e_1, \ldots, e_n$ have no common refinement.*

Proof. Assume that $(e_1, \ldots, e_n)$ is a state in M_{CR}, from which there is a path consisting of only must transitions leading to a state marked by Alg. 1 in the first repeat-loop. We

Input: A finite number n of deterministic WMTSs, $(d_i, D_i) \in d\mathcal{W}$, where $D_i = (S_i, \Sigma, \dashrightarrow_i, \longrightarrow_i, \delta_i)$ for $i = 1, \ldots, n$.
Output: The string *"No common refinement exists"* or a $(s_{CR}, M_{CR}) \in d\mathcal{W}$, where $M_{CR} = (S, \Sigma, \dashrightarrow, \longrightarrow, \delta)$ s.t. $(s_{CR}, M_{CR}) \leq_m (d_i, D_i)$ for all i.

begin
  $S := \emptyset$; $\longrightarrow := \emptyset$; $\dashrightarrow := \emptyset$; $Marked := \emptyset$; $Waiting := \{(d_1, \ldots, d_n)\}$;
  repeat
    Select $(e_1, \ldots, e_n) \in Waiting$; $Waiting := Waiting \setminus \{(e_1, \ldots, e_n)\}$;
    $S := S \cup \{(e_1, \ldots, e_n)\}$;
    forall the $a \in \Sigma$ **do**
      if $\exists i : e_i \xrightarrow{a}_i f_i$ *and* $\forall j : e_j \overset{a}{\dashrightarrow}_j f_j$ **then**
        $temp := \delta_1((e_1, a, f_1)) \cap \ldots \cap \delta_n((e_n, a, f_n))$;
        if $temp = \emptyset$ **then**
          $Marked := Marked \cup \{(e_1, \ldots, e_n)\}$;
        else
          $\longrightarrow := \longrightarrow \cup \left\{\left((e_1, \ldots, e_n), a, (f_1, \ldots, f_n)\right)\right\}$;
          $\delta\left(\left((e_1, \ldots, e_n), a, (f_1, \ldots, f_n)\right)\right) := temp$;
          if $(f_1, \ldots, f_n) \notin S$ **then** $Waiting := Waiting \cup \{(f_1, \ldots, f_n)\}$;
      if $\forall i : e_i \overset{a}{\dashrightarrow}_i f_i$ **then**
        $temp := \delta_1((e_1, a, f_1)) \cap \ldots \cap \delta_n((e_n, a, f_n))$;
        if $temp \neq \emptyset$ **then**
          $\dashrightarrow := \dashrightarrow \cup \left\{\left((e_1, \ldots, e_n), a, (f_1, \ldots, f_n)\right)\right\}$;
          $\delta\left(\left((e_1, \ldots, e_n), a, (f_1, \ldots, f_n)\right)\right) := temp$;
          if $(f_1, \ldots, f_n) \notin S$ **then** $Waiting := Waiting \cup \{(f_1, \ldots, f_n)\}$;
      if $\exists i : e_i \xrightarrow{a}_i f_i$ *and* $\exists j : e_j \not\overset{a}{\dashrightarrow}_j$ **then**
        $Marked := Marked \cup \{(e_1, \ldots, e_n)\}$;
  until $Waiting = \emptyset$;
  repeat
    $Marked' := Marked$;
    forall the $(e, a, f) \in \longrightarrow$ **do**
      if $f \in Marked$ **then** $Marked := Marked \cup \{e\}$;
  until $Marked' = Marked$;
  if $(d_1, \ldots, d_n) \in Marked$ **then return** *"No common refinement exists"*
  $S := S \setminus Marked$;
  forall the $(e, a, f) \in \longrightarrow \cup \dashrightarrow$ *where* $f \in Marked$ **do**
    $\longrightarrow := \longrightarrow \setminus \{(e, a, f)\}$; $\dashrightarrow := \dashrightarrow \setminus \{(e, a, f)\}$;
  $s_{CR} := (d_1, \ldots, d_n)$; **return** (s_{CR}, M_{CR})

Algorithm 1: Construction of the most general common refinement.

show that the states $e_1, \ldots, e_n$ have no common refinement.

First observe that if $(e_1, \ldots, e_n) \xrightarrow{a,V} (f_1, \ldots, f_n)$ exists in M_{CR} we know that $e_i \xrightarrow{a,V_i} f_i$, where $V \subseteq V_i$ for at least one i and $e_j \overset{a,V_j}{\dashrightarrow} f_j$, where $V \subseteq V_j$ for all j (Alg. 1, line 6-13). Let p be any common refinement of $e_1, \ldots, e_n$. Assume therefore that n refinement relations R_j exist such that $(p, e_j) \in R_j$ for all j. Then p must have a $p \xrightarrow{a,W} q$ transition, where $W \subseteq V_j$ and $(q, f_j) \in R_j$ for all j. The fact that q is a refinement of f_i is clear by the second item in Definition 3, since $(p, e_i) \in R_i$ and $e_i \xrightarrow{a,V_i} f_i$, where $V \subseteq V_i$, forces a $p \xrightarrow{a,W} q$ transition, where $W \subseteq V_i$ and $(q, f_i) \in R_i$. The fact that q is also required to be a refinement of $f_1, \ldots, f_n$ is given by the first item in Definition 3, since $(p, e_j) \in R_j$ for all j and $p \overset{a,W}{\dashrightarrow} q$. Since $e_1, \ldots, e_n$ are deterministic, $e_j \overset{a,V_j}{\dashrightarrow} f_j$, where $W \subseteq V_j$ for all j are forced to be the matching transitions, thus requiring $(q, f_j) \in R_j$ to hold.

Next observe that if a state $(g_1, \ldots, g_n) \in \mathit{Marked}$ in Alg. 1 then either $(g_1, \ldots, g_n)$ has been marked at line 9 or line 21. If it is marked at line 9, then there exists some $a \in \Sigma$ and $f_1, \ldots, f_n$ such that $g_i \xrightarrow{a}_i f_i$ for at least one i and $\delta_1((g_1, a, f_1)) \cap \ldots \cap \delta_n((g_n, a, f_n)) = \emptyset$. Otherwise (if it was marked at line 21) there exists i such that $g_i \xrightarrow{a}_i f_i$ and there exists j such that $g_j \not\overset{a}{\dashrightarrow}_j$. Both cases imply that $g_1, \ldots, g_n$ have no common refinement. The first case is obvious, since the weight set of the matching transition in a common refinement must be contained in every $\delta_i((g_i, a, f_i))$ (due to determinism), but no transition with an empty weight set is allowed. The second case also leads to no common implementation, since $g_i \xrightarrow{a,V_i} f_i$ forces $p \xrightarrow{a,V} q$, where $V \subseteq V_i$ in the refinement, but $p \overset{a,V}{\dashrightarrow} q$ cannot be matched from g_j.

These two observations lead to our conclusion, since any common refinement p of $e_1, \ldots, e_n$, with refinement relations R_j and $(p, e_j) \in R_j$ for all j eventually fulfills $(q, g_j) \in R_j$ for some q, because of the path consisting of only must transitions. However, since $g_1, \ldots, g_n$ cannot have a common refinement, R_j cannot exist. ■

By noting that Alg. 1 returns *"No common refinement exists"* only if there exists a path consisting of only must transitions from s_{CR} to a node marked before in the first repeat-loop and applying Lemma 8 to s_{CR} we have the following corollary.

Corollary 9. *If Alg. 1 returns "No common refinement exists" then the specifications $(d_1, D_1), \ldots, (d_n, D_n)$ have no common refinement.*

Lemma 10. *If $(d_1, D_1), \ldots, (d_n, D_n)$ have no common refinement then Alg. 1 returns "No common refinement exists".*

Proof. We prove the contraposition. That is, assume that Alg. 1 does not return *"No common refinement exists"*. This implies that $(d_1, \ldots, d_n)$ is not a marked state and that Alg. 1 returns (s_{CR}, M_{CR}). We want to construct relations $R_1, \ldots, R_n$ in order to show that s_{CR} is a common refinement of $d_1, \ldots, d_n$. We define the relations as

$$R_i = \{((e_1, \ldots, e_n), e_i) : (e_1, \ldots, e_n) \in M_{CR}\}$$

and continue to prove that these n relations fulfill the criteria of Definition 3. It is clear that $((d_1, \ldots, d_n), d_i) \in R_i$. Let $((e_1, \ldots, e_n), e_i) \in R_i$ and consider a must transition

$e_i \xrightarrow{a,V_i} f_i$. Since by assumption the algorithm returns (s_{CR}, M_{CR}) and $(e_1, \ldots, e_n) \in M_{CR}$, $(e_1, \ldots, e_n)$ is not marked. Therefore $e_j \overset{a,V_j}{\dashrightarrow} f_j$ exists for all j and the transition $(e_1, \ldots, e_n) \xrightarrow{a,\bigcap_j V_j} (f_1, \ldots, f_n)$ is added to M_{CR} in Alg. 1, line 11-13. Furthermore, $(f_1, \ldots, f_n)$ is not marked (and thus not removed), since otherwise $(e_1, \ldots, e_n)$ would have been marked during the repeat loop in line 23-27, a contradiction. Hence $(f_1, \ldots, f_n)$ is a state in M_{CR} and $((f_1, \ldots, f_n), f_i) \in R_i$ as required.

On the other hand, consider a may transition $(e_1, \ldots, e_n) \overset{a,V}{\dashrightarrow} (f_1, \ldots, f_n)$. By construction $e_i \overset{a,V_i}{\dashrightarrow} f_i$, where $V \subseteq V_i$ for all i. Hence $((f_1, \ldots, f_n), f_i) \in R_i$ for all i as required. ■

With the above lemmas and Alg. 1 we have the following complexity result.

Theorem 11. *The problem of existence of a common refinement for a given number of finite deterministic specifications is PSPACE-complete.*

Proof. In order to show containment in PSPACE, Cor. 9 and Lemma 10 give us that the existence of a common refinement is equivalent to the question whether the state s_{CR} gets marked by Alg. 1. In other words, this is equivalent to the question whether there is a must-path from s_{CR} to some state marked directly in line 9 or 21 of the algorithm. Such a path can be nondeterministically guessed on the fly (without constructing the whole state-space) and by Savitch's theorem this implies the containment in PSPACE.

The hardness result follows directly from PSPACE-hardness of the common implementation problem for unweighted MTSs shown in [BKLS09b]. ■

The last theorem states that M_{CR} is the largest common refinement.

Theorem 12 (Maximality of M_{CR})**.** *If Alg. 1 returns (s_{CR}, M_{CR}) then for every $(t, N) \in \mathcal{W}$ such that $(t, N) \leq_m (d_i, D_i)$ for all i, $1 \leq i \leq n$, it holds that $(t, N) \leq_m (s_{CR}, M_{CR})$.*

Proof. Let $(t, N) \in \mathcal{W}$ be a common refinement of $(d_1, D_1), \ldots, (d_n, D_n)$. This means that there exist n relations $R_1, \ldots, R_n$ satisfying the conditions in Definition 3. We construct a new relation Q satisfying the same conditions in order to prove that $(t, N) \leq_m (s_{CR}, M_{CR})$. The relation Q is defined as follows:

$$(s, (e_1, \ldots, e_n)) \in Q \text{ if and only if } (s, e_i) \in R_i \text{ for all } i,\ 1 \leq i \leq n.$$

We observe that $(t, (d_1, \ldots, d_n)) \in Q$, since $(t, d_i) \in R_i$ for all i. This satisfies the first condition in the refinement definition. Let now $(s, (e_1, \ldots, e_n)) \in Q$ and consider what happens in case of must and may transitions.

Consider a transition $s \overset{a,V}{\dashrightarrow} s'$. Then for all i we have that for all e_i such that $(s, e_i) \in R_i$ there exists f_i such that $e_i \overset{a,W_i}{\dashrightarrow} f_i$ with $V \subseteq W_i$ and $(s', f_i) \in R_i$. This implies that $q = (e_1, \ldots, e_n) \overset{a,W_1 \cap \ldots \cap W_n}{\dashrightarrow} (f_1, \ldots, f_n)$ is a transition in M_{CR}. Since $V \subseteq W_1 \cap \ldots \cap W_n$ and $(s', f_i) \in R_i$ for all i, then $(s', (f_1, \ldots, f_n)) \in Q$ as desired. Notice that $(f_1, \ldots, f_n)$ cannot be a node which was removed in Alg. 1 since this would imply that $(f_1, \ldots, f_n)$ is

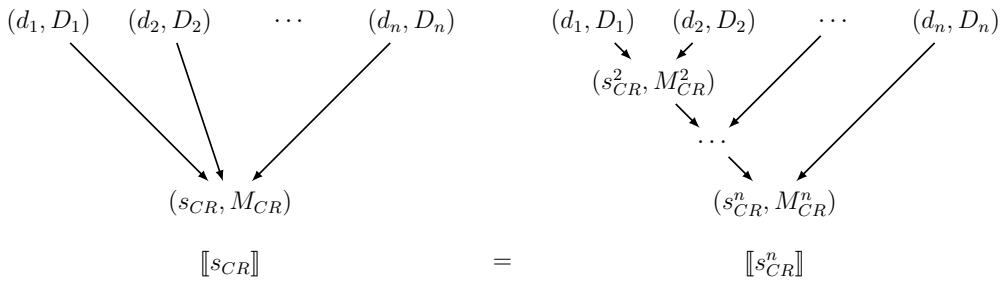


Figure 4.: Two ways of using Alg. 1 yielding the same result.

marked. The states $f_1, \ldots, f_n$ would not therefore have a common refinement by Lemma 8. This contradicts the fact that $(s', f_i) \in R_i$ for all i.

Consider now a transition $(e_1, \ldots, e_n) \xrightarrow{a,V} (f_1, \ldots, f_n)$. By construction of M_{CR} we know that there exists at least one e_j such that $e_j \xrightarrow{a,W_j} f_j$ and that $e_i \overset{a,W_i}{\dashrightarrow} f_i$ exist for all i (Alg. 1, line 6-13). Since $(d_1, D_1), \ldots, (d_n, D_n)$ are deterministic, f_i (for all i) are unique. Now because $e_j \xrightarrow{a,W_j} f_j$ and $(s, e_j) \in R_j$, we get that $s \xrightarrow{a,W} s'$ with $W \subseteq W_j$ such that $(s', f_j) \in R_j$. This, however, also means that $s \overset{a,W}{\dashrightarrow} s'$ and because $(s, e_i) \in R_i$ for all i, we get that $(s', f_i) \in R_i$ and $W \subseteq W_i$ for all i. Notice that $V = W_1 \cap \ldots \cap W_n$, so $W \subseteq V$ holds. Thus $(s', (f_1, \ldots, f_n)) \in Q$ by the definition of Q. ■

As a corollary and due to the transitivity of $\leq_m$ (Lemma 5), Cor. 9, Lemma 10 and the maximality of M_{CR} (Thm. 12) we get the main result.

Corollary 13. *Let $(d_1, D_1), \ldots, (d_n, D_n)$ be finite deterministic WMTSs and assume that Alg. 1 returns a specification (s_{CR}, M_{CR}). A specification $(s, M) \in \mathcal{W}$ is a common refinement of the specifications $(d_1, D_1), \ldots, (d_n, D_n)$ if and only if $(s, M) \leq_m (s_{CR}, M_{CR})$.*

This theorem allows us to find a common refinement (and thus also a common implementation) in a stepwise and iterative manner. Consider Figure 4. Here (s_{CR}, M_{CR}) is constructed by giving $(d_1, D_1), \ldots, (d_n, D_n) \in d\mathcal{W}$ as input to Alg. 1. The specification (s^n_{CR}, M^n_{CR}) is, on the other hand, constructed by using Alg. 1 iteratively. First $(d_1, D_1), (d_2, D_2)$ is given as input and then the output, (s^2_{CR}, M^2_{CR}), and (d_3, D_3) is used as input, continuing in this way until the last received output and (d_n, D_n) is given as input, finally outputting (s^n_{CR}, M^n_{CR}). The theorem below states that both applications of the algorithm lead to the same set of possible implementations.

Theorem 14. *Let $(d_1, D_1), \ldots, (d_n, D_n)$ be finite deterministic WMTSs and assume that (s_{CR}, M_{CR}) and (s^n_{CR}, M^n_{CR}) are the specifications obtained as illustrated in Figure 4. Then $[\![s_{CR}]\!] = [\![s^n_{CR}]\!]$.*

Proof. Using Corollary 13 gives us that $(s^n_{CR}, M^n_{CR}) \leq_m (s_{CR}, M_{CR})$. The other direction is an easy induction in n, the number of deterministic specifications. As a base case we have $n = 2$. The two uses of Alg. 1 are here equal, and the theorem follows. For the induction step assume $(s_{CR}, M_{CR}) \leq_m (s^{n-1}_{CR}, M^{n-1}_{CR})$ holds. Notice that now

$(s_{CR}, M_{CR}) \leq_m (s^n_{CR}, M^n_{CR})$ also holds, since (s^n_{CR}, M^n_{CR}) is the output when Alg. 1 is given $(s^{n-1}_{CR}, M^{n-1}_{CR})$ and (d_n, D_n) as input and Corollary 13 is thus applicable. Lemma 6 now implies $[\![s_{CR}]\!] = [\![s^n_{CR}]\!]$. ■

This result is more general than the algorithm in [BKLS09b], which checks only for the existence of a common implementation of (unweighted) modal specifications. The algorithm presented here furthermore constructs the most permissive common refinement and provides support for step-wise development of systems.

4. Logical Characterisation

In the previous section we discussed algorithms for constructing some largest common refinement for a given set of weighted deterministic modal specifications. Now we shall turn our attention to a logical characterisation of weighted modal transition systems. We define an extension of the well-known CTL logic [EC80] that will allow us to state logical queries that include constraints about the weights along the finite and infinite paths. There are several well justified choices for the definition of the constraints on the paths. We provide a meta-definition of a general constraint form which specializes to many useful constraint choices. Our main result is that as long as a certain monotonicity property is preserved, a specification satisfies a given logical formula if and only if all its refinements do. This result can be understood as a soundness principle for the suggested logic.

We start with the definition of must-/may-paths in weighted modal transition systems.

Definition 15 (Path)**.** A *must-path* in a WMTS $M = (S, \Sigma, \dashrightarrow, \longrightarrow, \delta)$ is a finite or infinite sequence π of transitions of the form

$$\pi = s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \dots \ .$$

A must-path is *maximal* if it is infinite or it ends in a state with no outgoing may transitions (and hence of course also no outgoing must transitions). The set of all maximal must-paths starting from a state s is denoted by $\mathsf{maxmustP}(s)$.

Similarly, a *may-path* is a finite or infinite sequence π of transitions of the form

$$\pi = s_1 \overset{a_1}{\dashrightarrow} s_2 \overset{a_2}{\dashrightarrow} s_3 \overset{a_3}{\dashrightarrow} \dots \ .$$

A may-path is *maximal* if it is infinite or it ends in a state with no outgoing must transition (note that outgoing may transitions are allowed). The set of all maximal may-paths starting from a state s is denoted by $\mathsf{maxmayP}(s)$. Notice that a must-path is not necessarily a prefix of a maximal must-path and that a maximal may-path may be a strict prefix of another maximal may-path.

Given a must- or may-path in the form above, the notation $\pi[j]$ denotes the j'th state of the path, that is $\pi[j] = s_j$.

For specifying logical properties we suggest a notion of weighted action-based CTL (WCTL), a particular extension of CTL (see e.g. [BK08]). The action-based syntax is based on the work of De Nicola and Vaandrager [NV90] which introduces an action-labelled next operator. In [NV90] they discuss a close relationship between action-based and state-based logics (see also [MKB08]). We further extend their logic such that it can be interpreted over modal transition systems and we add a generic weight constraint function in order to reason about the cost of the transitions and demonstrate a few examples of well-justified weight constraint functions.

Let us first define the so-called *action formulae*:

$$\chi, \chi' ::= \mathsf{true} \mid \quad a \quad \mid \neg\chi \mid \chi \wedge \chi'$$

where $a \in \Sigma$ ranges over the actions of a given WMTS. The semantics to action formulae is given by the following satisfaction relation ($a, b \in \Sigma$):

$$\begin{aligned} a &\models \mathsf{true} \\ a &\models b \qquad \text{iff} \quad a = b \\ a &\models \neg\chi \qquad \text{iff} \quad a \not\models \chi \\ a &\models \chi \wedge \chi' \quad \text{iff} \quad a \models \chi \text{ and } a \models \chi' \quad . \end{aligned}$$

The *(state) formulae* of WCTL are now generated by the following abstract syntax:

$$\begin{aligned} \varphi, \varphi_1, \varphi_2 ::= \mathsf{true} \mid \mathsf{false} \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \mathsf{EX}^{\mathsf{c}}_{\chi}\, \varphi \mid \mathsf{AX}^{\mathsf{c}}_{\chi}\, \varphi \\ \mid \mathsf{E}\,(\varphi_1\, \mathsf{U}^{\mathsf{c}}_{\chi}\, \varphi_2) \mid \mathsf{A}\,(\varphi_1\, \mathsf{U}^{\mathsf{c}}_{\chi}\, \varphi_2) \mid \mathsf{E}\,(\varphi_1\, \mathsf{R}^{\mathsf{c}}_{\chi}\, \varphi_2) \mid \mathsf{A}\,(\varphi_1\, \mathsf{R}^{\mathsf{c}}_{\chi}\, \varphi_2) \end{aligned}$$

where χ ranges over action formulae and $\mathsf{c} : \mathcal{I}^* \cup \mathcal{I}^\omega \to \{0, 1\}$ is a constraint function assigning 0 (false) or 1 (true) to any finite and infinite sequence of weight intervals (for the definition of $\mathcal{I}$ see the first paragraph of Section 2). We moreover require that c satisfies the following monotonicity property:

$$\text{if } \mathsf{c}(w_1, w_2, \ldots) = 1 \text{ then } \mathsf{c}(w_1', w_2', \ldots) = 1 \text{ for any } w_i' \subseteq w_i \text{ for all } i.$$

This means that if some sequence of intervals is acceptable by the constraint function, so will be the sequence containing any subintervals. By $\mathcal{L}$ we denote the set of all WCTL formulae.

This syntax is similar to the standard action-based CTL. The main difference is the superscript 'c' attached to the next, until and release operators. For example, $\mathsf{E}\,(\varphi_1\, \mathsf{U}^{\mathsf{c}}_{\chi}\, \varphi_2)$ holds in a state s if there exists a maximal must-path which satisfies that φ_2 holds in some state along the path, φ_1 holds in all states prior to that state, the actions on the subpath where φ_1 holds satisfy χ and c is true for the sequence of intervals belonging to the subpath where φ_1 holds. The reason for choosing a must-path is that the existence of such a path in the specification will guarantee its existence also in any of its refinements. Similarly, the formula $\mathsf{E}\,(\varphi_1\, \mathsf{R}^{\mathsf{c}}_{\chi}\, \varphi_2)$ holds in a state s if there exists a maximal must-path which satisfies that φ_2 holds in all states along the path, a requirement that is dropped

as soon as φ_1 holds, the actions on the path where φ_2 holds satisfy χ and c is true for the sequence of intervals belonging to the path where φ_2 holds (hence the need for the constraint function to be defined over infinite sequences of intervals too). For the path quantifier A, the temporal operators U and R have a similar meaning, only in this case we require that all maximal may-paths satisfy these properties.

The semantics of WCTL formulae is then interpreted over the states of a WMTS. Let $M = (S, \Sigma, \dashrightarrow, \longrightarrow, \delta) \in \mathcal{W}$, χ range over action formulae, $s \in S$, and φ, φ_1 and φ_2 be formulae from $\mathcal{L}$. The satisfaction relation $\models$ is defined inductively by $s \models \mathsf{true}$, $s \not\models \mathsf{false}$ and

$$
\begin{array}{lll}
s \models \varphi_1 \wedge \varphi_2 & \text{iff} & s \models \varphi_1 \text{ and } s \models \varphi_2 \\
s \models \varphi_1 \vee \varphi_2 & \text{iff} & s \models \varphi_1 \text{ or } s \models \varphi_2 \\
s \models \mathsf{EX}^{\mathsf{c}}_{\chi}\, \varphi & \text{iff} & \exists (s \xrightarrow{a} s') : a \models \chi \ \wedge\ s' \models \varphi \ \wedge\ \mathsf{c}(\delta(s,a,s')) = 1 \\
s \models \mathsf{AX}^{\mathsf{c}}_{\chi}\, \varphi & \text{iff} & \forall (s \overset{a}{\dashrightarrow} s') \text{ where } a \models \chi : s' \models \varphi \ \wedge\ \mathsf{c}(\delta(s,a,s')) = 1 \\
s \models \mathsf{E}\,(\varphi_1\, \mathsf{U}^{\mathsf{c}}_{\chi}\, \varphi_2) & \text{iff} & \exists \pi = s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots \in \mathsf{maxmustP}(s) : \\
 & & \exists i \geq 1 : s_i \models \varphi_2 \\
 & & \wedge\ \forall j \in \{1, \ldots, i-1\} : (s_j \models \varphi_1 \ \wedge\ a_j \models \chi) \\
 & & \wedge\ \mathsf{c}(\delta(s_1,a_1,s_2), \delta(s_2,a_2,s_3), \ldots, \delta(s_{i-1},a_{i-1},s_i)) = 1 \\
s \models \mathsf{A}\,(\varphi_1\, \mathsf{U}^{\mathsf{c}}_{\chi}\, \varphi_2) & \text{iff} & \forall \pi = s_1 \overset{a_1}{\dashrightarrow} s_2 \overset{a_2}{\dashrightarrow} \ldots \in \mathsf{maxmayP}(s) \\
 & & \text{where } a_i \models \chi \text{ for all } i : \\
 & & \exists i \geq 1 : s_i \models \varphi_2 \ \wedge\ (\forall j \in \{1, \ldots, i-1\} : s_j \models \varphi_1) \\
 & & \wedge\ \mathsf{c}(\delta(s_1,a_1,s_2), \delta(s_2,a_2,s_3), \ldots, \delta(s_{i-1},a_{i-1},s_i)) = 1 \\
s \models \mathsf{E}\,(\varphi_1\, \mathsf{R}^{\mathsf{c}}_{\chi}\, \varphi_2) & \text{iff} & \exists \pi = s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots \in \mathsf{maxmustP}(s) : \\
 & & \Big(\forall k \geq 1 : s_k \models \varphi_2 \ \wedge\ a_k \models \chi \\
 & & \wedge\ \mathsf{c}(\delta(s_1,a_1,s_2), \delta(s_2,a_2,s_3), \ldots) = 1\Big) \vee \\
 & & \Big(\exists i \geq 1 : s_i \models \varphi_1 \wedge \forall j \in \{1, \ldots, i\} : (s_j \models \varphi_2 \wedge a_j \models \chi) \\
 & & \wedge\, \mathsf{c}(\delta(s_1,a_1,s_2), \delta(s_2,a_2,s_3), \ldots, \delta(s_{i-1},a_{i-1},s_i)) = 1\Big) \\
s \models \mathsf{A}\,(\varphi_1\, \mathsf{R}^{\mathsf{c}}_{\chi}\, \varphi_2) & \text{iff} & \forall \pi = s_1 \overset{a_1}{\dashrightarrow} s_2 \overset{a_2}{\dashrightarrow} \ldots \in \mathsf{maxmayP}(s) \\
 & & \text{where } a_i \models \chi \text{ for all } i : \\
 & & \Big(\forall k \geq 1 : s_k \models \varphi_2 \\
 & & \wedge\ \mathsf{c}(\delta(s_1,a_1,s_2), \delta(s_2,a_2,s_3), \ldots) = 1\Big) \vee \\
 & & \Big(\exists i \geq 1 : s_i \models \varphi_1 \ \wedge\ (\forall j \in \{1, \ldots, i\} : s_j \models \varphi_2) \\
 & & \wedge\, \mathsf{c}(\delta(s_1,a_1,s_2), \delta(s_2,a_2,s_3), \ldots, \delta(s_{i-1},a_{i-1},s_i)) = 1\Big).
\end{array}
$$

If the system M is not clear from the context, we also use the notation $(s, M) \models \varphi$ meaning that $s \models \varphi$ where s a state in M. We remark that for the cases of $\mathsf{E}\ (\varphi_1\ \mathsf{U}^{\mathsf{c}}_{\chi}\ \varphi_2)$ and $\mathsf{A}\ (\varphi_1\ \mathsf{R}^{\mathsf{c}}_{\chi}\ \varphi_2)$ we may consider also paths that are not necessarily maximal, but for the sake of techincal conveniences we restrict ourself to maximal runs in all cases.

Notice that, as usual, we can express the temporal modalities 'eventually' ($\mathsf{F}^{\mathsf{c}}_{\chi}$) and 'always' ($\mathsf{G}^{\mathsf{c}}_{\chi}$).

$$\begin{aligned}
\mathsf{EF}^{\mathsf{c}}_{\chi}\ \varphi &\equiv \mathsf{E}\ (\mathsf{true}\ \mathsf{U}^{\mathsf{c}}_{\chi}\ \varphi)\\
\mathsf{AF}^{\mathsf{c}}_{\chi}\ \varphi &\equiv \mathsf{A}\ (\mathsf{true}\ \mathsf{U}^{\mathsf{c}}_{\chi}\ \varphi)\\
\mathsf{EG}^{\mathsf{c}}_{\chi}\ \varphi &\equiv \mathsf{E}\ (\mathsf{false}\ \mathsf{R}^{\mathsf{c}}_{\chi}\ \varphi)\\
\mathsf{AG}^{\mathsf{c}}_{\chi}\ \varphi &\equiv \mathsf{A}\ (\mathsf{false}\ \mathsf{R}^{\mathsf{c}}_{\chi}\ \varphi)
\end{aligned}$$

The generic definition of the constraint function c can be specialized in order to express a variety of interesting properties that we want our sequences of intervals to fulfil. We will now give two examples.

Example 16. Consider the ATM machines shown in Figure 1 in Section 1. It might be worth knowing, for example, whether or not in any implementation it is possible to check the balance without entering PIN while consuming between 0 and 15 units of power. We thus want to reason about the accumulated energy along a given path. For this purpose, let the constraint function c_S be given as

$$\mathsf{c}_S(w_1, w_2, \ldots) = \begin{cases} 1 & \text{if } \left[\sum_{i=1} n_i, \sum_{i=1} m_i\right] \subseteq S \\ 0 & \text{otherwise,} \end{cases}$$

where $w_i = [n_i, m_i]$ and S is a given set of elements from $\mathbb{Z} \cup \{-\infty, \infty\}$. The constraint function then returns 1 if the interval containing all possible sums of weights belonging to each interval is contained in the set S, and 0 otherwise.

Notice that if $\mathsf{c}(w_1, w_2, \ldots) = 1$ then $\mathsf{c}(w'_1, w'_2, \ldots) = 1$ for any $w'_i \subseteq w_i$ for all i, since smaller intervals do not give rise to any new accumulated sums, thus still preserving the requirement of being a subset of S.

Using this constraint function one can specify that every possible total accumulated weight along some path should be contained in $[0, 15]$, as required in the ATM machine. The WCTL formula to check this is

$$\varphi \equiv \mathsf{E}\left(\mathsf{true}\ \mathsf{U}^{\mathsf{c}[0,15]}_{\neg \mathrm{PIN}}\ \left(\ \mathsf{EX}^{\mathsf{c}[-\infty,\infty]}_{\mathrm{return}}\ \mathsf{true}\right)\right) \equiv \mathsf{EF}^{\mathsf{c}[0,15]}_{\neg \mathrm{PIN}}\ \left(\ \mathsf{EX}^{\mathsf{c}[-\infty,\infty]}_{\mathrm{return}}\ \mathsf{true}\right),$$

stating that there exists a path where the action 'return' is enabled in some state in the future (with an arbitrary cost), and that we until reaching the state enabling 'return' must not take a transition with action 'PIN' and that the accumulated cost on this subpath (where the balance is checked) must consume between 0 and 15 power units.

Consulting the leftmost specification in Figure 1 we see that $s_1 \models \varphi$ holds, since a must-path $s_1 \overset{\mathrm{card},[2,5]}{\longrightarrow} s_2 \overset{\mathrm{balance},[1,6]}{\longrightarrow} s_3$ exists, and from s_3 an outgoing must transition with

the action 'return' is required. Any implementation would therefore also require these three transitions.

On the other hand, $t_1 \not\models \varphi$, since in the specification the 'balance' transition is only optional, and thus might not be present in an implementation.

Example 17. As another example consider a specification of a gas tank, able to both gain and lose gas. In this case we are interested in keeping the volume of gas in the tank between some interval at all times, since the tank would otherwise explode or have a too low volume. It is therefore not adequate to only consider the volume at the end of a given path. All subpaths must also fulfil the interval bound (note that we allow for negative weights). We therefore define c_S as

$$\mathsf{c}_S(w_1, w_2, \ldots) = \begin{cases} 1 & \text{if } \forall j \geq 1 : \left[\sum_{i=1}^{j} n_i, \sum_{i=i}^{j} m_i\right] \subseteq S \\ 0 & \text{otherwise,} \end{cases}$$

where $w_i = [n_i, m_i]$ and S is a given set of elements from $\mathbb{Z} \cup \{-\infty, \infty\}$. In this way we specify, using the operator G, that the volume of the tank must be between 0 and 100 everywhere along all potentially infinite paths where the action emergency is not taken by

$$\mathsf{AG}^{c_{[0,100]}}_{\neg \text{emergency}}\ \mathsf{true}\ .$$

If emergency is triggered along some path, the tank will shut down and the volume need not be guarded any more.

This kind of weight constraint function proved useful e.g. in [BFL+08] where the existence of such an infinite path was studied in the context of weighted timed automata. Observe again that if $\mathsf{c}(w_1, w_2, \ldots) = 1$ then $\mathsf{c}(w'_1, w'_2, \ldots) = 1$ for any $w'_i \subseteq w_i$ for all i.

We shall now formulate a technical lemma that will be used in the proof of the main theorem of this section.

Lemma 18. *Let $(s_1, M), (t_1, N) \in \mathcal{W}$ and $(s_1, M) \leq_m (t_1, N)$.*

1. *If $\pi_N \in \mathsf{maxmustP}(t_1)$ then there exists $\pi_M \in \mathsf{maxmustP}(s_1)$ of the same length as π_N, such that $(\pi_M[i], M) \leq_m (\pi_N[i], N)$ for all i.*

2. *If $\pi_M \in \mathsf{maxmayP}(s_1)$ then there exists $\pi_N \in \mathsf{maxmayP}(t_1)$ of the same lento as π_M, such that $(\pi_M[i], M) \leq_m (\pi_N[i], N)$ for all i.*

Moreover, in both cases the weight intervals on the path π_M are subintervals of the corresponding intervals on the path π_N.

Proof. Let $(s_1, M), (t_1, N) \in \mathcal{W}$ and assume that $s_1 \leq_m t_1$.

1. Let $\pi_N = t_1 \xrightarrow{a_1,V_1} t_2 \xrightarrow{a_2,V_2} t_3 \xrightarrow{a_3,V_3} \ldots$ be a maximal must-path in N. We want to show that there exists a maximal must-path $\pi_M = s_1 \xrightarrow{a_1,W_1} s_2 \xrightarrow{a_2,W_2} s_3 \xrightarrow{a_3,W_3} \ldots$ in M such that $s_i \leq_m t_i$ and $W_i \subseteq V_i$ for all i. We prove this by induction. By induction hypothesis we assume that there is a path $s_1 \xrightarrow{a_1,W_1} s_2 \xrightarrow{a_2,W_2} \ldots \xrightarrow{a_{j-1},W_{j-1}} s_j$ in M for $j > 1$ such that

$s_i \leq_m t_i$ and $W_i \subseteq V_i$ for all i, $1 \leq i \leq j$. (The base case where $j = 1$ requires that $s_1 \leq_m t_1$ which is true by the assumption of the lemma.) We now distinguish two cases (recall that π_N is a maximal must-path).

- *Case where t_j has no outgoing may transitions.* Because $s_j \leq_m t_j$ we get that s_j cannot have any outgoing transitions either and a maximal must-path in N was matched by a maximal must-path in M as required.
- *Case where $t_j \xrightarrow{a_j,V_j} t_{j+1}$.* Because $s_j \leq_m t_j$ there must be a transition $s_j \xrightarrow{a_j,W_j} s_{j+1}$ such that $s_{j+1} \leq_m t_{j+1}$ and $W_j \subseteq V_j$. Hence there is a path $s_1 \xrightarrow{a_1,W_1} s_2 \xrightarrow{a_2,W_2} \ldots \xrightarrow{a_j,W_j} s_{j+1}$ in M such that $s_i \leq_m t_i$ and $W_i \subseteq V_i$ for all i, $1 \leq i \leq j+1$, as required by the induction.

2. Let $\pi_M \in \mathsf{maxmayP}(s_1)$ be a maximal may-path in M. We want to find a matching maximal may-path in N. The arguments are symmetric as in the proof of part 1. Notice that the maximality of π_M in case of a finite path, i.e. the absence of any must transition at the end of the path, implies that the path π_N constructed in a similar manner as in part 1. is also maximal (the presence of a must transition at its last state would enforce the presence of a must transition in the last state of π_M). ■

Given a specification $(s, M) \in \mathcal{W}$ and a WCTL formula φ such that $(s, M) \models \varphi$, the following theorem shows that any refinement also satisfies φ. This problem is closely related to generalized model checking (as defined in [BG00]), which asks, given $(s, M) \in \mathcal{W}$ and $\varphi \in \mathcal{L}$, does there exists an implementation $(i, I) \in [\![s]\!]$, such that $(i, I) \models \varphi$. In our case we, on the other hand, consider the validity problem for all refinements—does any refinement fulfil φ?

Theorem 19. *Let $(t, N) \in \mathcal{W}$ and let $\varphi \in \mathcal{L}$ be a WCTL formula. Then $(t, N) \models \varphi$ if and only if $(s, M) \models \varphi$ for all (s, M) s.t. $(s, M) \leq_m (t, N)$.*

Proof. The 'if' part is trivial since (t, N) is a refinement of itself. We prove 'only if' below.

Let $(s, M), (t, N) \in \mathcal{W}$ be two weighted modal transition systems, where $M = (S_M, \Sigma, \dashrightarrow, \longrightarrow, \delta_M)$ and $N = (S_N, \Sigma, \dashrightarrow, \longrightarrow, \delta_N)$. We show that for any formula $\varphi \in \mathcal{L}$:

$$\text{if } (s, M) \leq_m (t, N) \text{ and } (t, N) \models \varphi \text{ then } (s, M) \models \varphi \ . \tag{1}$$

The proof is by structural induction over the structure of the formula φ.
Induction basis: The cases $\varphi = \mathsf{true}$ and $\varphi = \mathsf{false}$ are trivial.
Induction step: Assume φ_1 and φ_2 are state formulae for which (1) hold.

- $\varphi = \varphi_1 \wedge \varphi_2$:
  Since the induction hypothesis applies to φ_1 and φ_2 we have

$$t \models \varphi_1 \wedge \varphi_2 \implies (t \models \varphi_1) \text{ and } (t \models \varphi_2) \implies$$
$$(s \models \varphi_1) \text{ and } (s \models \varphi_2) \implies s \models \varphi_1 \wedge \varphi_2.$$

- $\varphi = \varphi_1 \vee \varphi_2$:

  Similar as for conjunction.

- $\varphi = \mathsf{EX}^{\mathsf{c}}_{\chi}\ \varphi_1$:

  If $t \models \varphi$ then there exists $t \xrightarrow{a,W} t'$, where $a \models \chi$, $t' \models \varphi_1$ and $\mathsf{c}(W) = 1$. Since $s \leq_m t$, we conclude that $s \xrightarrow{a,V} s'$, where $V \subseteq W$ exists such that $s' \leq_m t'$. The requirement on the constraint function c implies that also $\mathsf{c}(V) = 1$, since $V \subseteq W$. By the induction hypothesis $s' \models \varphi_1$ and thus $s \models \varphi$ as well.

- $\varphi = \mathsf{E}\ (\varphi_1\ \mathsf{U}^{\mathsf{c}}_{\chi}\ \varphi_2)$:

  If $t \models \varphi$ then there exists a must-path $\pi_N = t_1 \xrightarrow{a_1} t_2 \xrightarrow{a_2} \ldots$ with $t_1 = t$ in $\mathsf{maxmustP}(t)$ on which there exists j such that $t_j \models \varphi_2$, $t_k \models \varphi_1$ and $a_k \models \chi$ for all $k < j$ and $\mathsf{c}(\delta(t_1 \xrightarrow{a_1} t_2), \delta(t_2 \xrightarrow{a_2} t_3), \ldots, \delta(t_{j-1} \xrightarrow{a_{j-1}} t_j)) = 1$. By Lemma 18 we know that there exists a must-path $\pi_M = s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots$ with $s_1 = s$ in $\mathsf{maxmustP}(s)$ such that $s_i \leq_m t_i$ for all i. The induction hypothesis implies that also $s_j \models \varphi_2$ and that $s_k \models \varphi_1$ for all $k < j$. By the requirement on c we also have that if $\mathsf{c}(\delta(t_1 \xrightarrow{a_1} t_2), \delta(t_2 \xrightarrow{a_2} t_3), \ldots, \delta(t_{j-1} \xrightarrow{a_{j-1}} t_j)) = 1$ then $\mathsf{c}(\delta(s_1 \xrightarrow{a_1} s_2), \delta(s_2 \xrightarrow{a_2} s_3), \ldots, \delta(s_{j-1} \xrightarrow{a_{j-1}} s_j)) = 1$, since $\delta(s_i \xrightarrow{a_i} s_{i+1}) \subseteq \delta(t_i \xrightarrow{a_i} t_{i+1})$ for all i. Hence $s \models \varphi$.

- $\varphi = \mathsf{E}\ (\varphi_1\ \mathsf{R}^{\mathsf{c}}_{\chi}\ \varphi_2)$:

  The same reasoning as in the previous case is used. As $t \models \varphi$ then there exists a must-path $\pi_N \in \mathsf{maxmustP}(t)$ such that φ_2 and χ holds in all states and transitions respectively until and including the first state where φ_1 holds (possibly never) and c equals 1 when evaluated on the weight intervals corresponding to the path where φ_2 holds. Again Lemma 18 provides a must-path $\pi_M \in \mathsf{maxmustP}(s)$ such that the induction hypothesis and the requirement on c gives us that $s \models \varphi$.

- $\varphi = \mathsf{AX}^{\mathsf{c}}_{\chi}\ \varphi_1$:

  Consider here an arbitrary transition $s \overset{a,V}{\dashrightarrow} s'$ where $a \models \chi$. Since $s \leq_m t$, there exists $t \overset{a,W}{\dashrightarrow} t'$ with $V \subseteq W$ such that $s' \leq_m t'$. As $t \models \varphi$, we know that $t' \models \varphi_1$ and $\mathsf{c}(W) = 1$. By the induction hypothesis and the requirement on c, we get that also $s \models \varphi$.

- $\varphi = \mathsf{A}\ (\varphi_1\ \mathsf{U}^{\mathsf{c}}_{\chi}\ \varphi_2)$:

  Consider an arbitrary path $\pi_M = s_1 \overset{a_1}{\dashrightarrow} s_2 \overset{a_2}{\dashrightarrow} \ldots$ with $s_1 = s$ in $\mathsf{maxmayP}(s)$ where $a_i \models \chi$ for all i. By Lemma 18 a path $\pi_N = t_1 \overset{a_1}{\dashrightarrow} t_2 \overset{a_2}{\dashrightarrow} \ldots$ with $t_1 = t$ in $\mathsf{maxmayP}(t)$ exists such that $s_i \leq_m t_i$ for all i. As $t \models \varphi$, then there exists j such that t_j satisfies φ_2, $t_k \models \varphi_1$ for all $k < j$ and $\mathsf{c}(\delta(t_1 \overset{a_1}{\dashrightarrow} t_2), \delta(t_2 \overset{a_2}{\dashrightarrow} t_3), \ldots, \delta(t_{j-1} \overset{a_{j-1}}{\dashrightarrow} t_j)) = 1$. By the induction hypothesis $s_j \models \varphi_2$ and $s_k \models \varphi_1$

for all $k < j$. Since $\delta(s_i \overset{a_i}{\dashrightarrow} s_{i+1}) \subseteq \delta(t_i \overset{a_i}{\dashrightarrow} t_{i+1})$ for all i, this implies that $\mathsf{c}(\delta(s_1 \overset{a_1}{\dashrightarrow} s_2), \delta(s_2 \overset{a_2}{\dashrightarrow} s_3), \ldots, \delta(t_{j-1} \overset{a_{j-1}}{\dashrightarrow} t_j)) = 1$. We now have that $s \models \varphi$ as required.

- $\varphi = \mathsf{A}\ (\varphi_1\ \mathsf{R}^{\mathsf{c}}_{\chi}\ \varphi_2)$:

  Again, we prove that if $t \models \varphi$ then also $s \models \varphi$. This is done as in the previous case by considering an arbitrary path in $\mathsf{maxmayP}(s)$ where all actions satisfy χ and applying Lemma 18, the induction hypothesis and the requirement on c.

■

The reader may wonder why this action-based CTL is in positive normal form. The reason for this is that we require that a formula satisfied by a specification is also satisfied by all refinements. This does not hold for a formula of the form $\neg\varphi$. Consider for instance the specification consisting of two states s_1 and s_2, with $s_1 \overset{a,W}{\dashrightarrow} s_2$. Then the state s_1 satisfies $\varphi \equiv \neg\ \mathsf{EX}^{\mathsf{c}}_{a}\ \mathsf{true}$, with c returning constantly 1, since EX requires a must transition. However, in the refinement consisting of two states i_1 and i_2 with $i_1 \xrightarrow{a,V} i_2$, $V \subseteq W$, the state i_1 does not satisfy φ, since there indeed exists a must transition from s_1 with the action a. On the other hand, in the refinement consisting of an isolated state j_1 with no transitions $j_1 \models \varphi$ holds.

Thus, showing that a specification does not satisfy some φ only implies the existence of at least one refinement satisfying $\neg\varphi$, not all of them as required by the modal refinement methodology.

5. Conclusion and Future Work

We presented a novel extension of modal transition systems called weighted modal transition systems where each transition is decorated with an interval of weights, describing all possible values that can be used in an implementation. Furthermore we constructed the largest common refinement of a number of finite deterministic specifications, and proved the correctness of the construction. This result generalizes the previously known algorithm for the common implementation problem on unweighted deterministic modal transition systems. We also suggested a notion of weighted CTL logic in order to reason about the properties of the weighted modal transition systems and argued for the soundness of this choice.

Clearly the proposed logic is undecible due to its generality. As a future work it would therefore be interesting to identify decidable fragments of the logic. This can be achieved e.g. by considering a subset of the allowed state formulae, an unweighted version of the logic or by reasoning only about implementations.

In our future work we will also consider the common implementation/specification problems of nondeterministic specifications, a determinisation construction, algorithmic aspects of the generalized model checking problem for weighted modal specifications as well as the extension of the formalisms to mixed systems where the must transitions

are not necessarily included in the may transitions. One might also consider a lattice of values as weight domain instead of the less general intervals considered here.

D Extending Modal Transition Systems with Structured Labels

Sebastian Bauer
Institut für Informatik, Ludwig-Maximilians-Universität München, Germany

Line Juhl Kim G. Larsen
Aalborg University, Department of Computer Science, Denmark

Axel Legay
INRIA/IRISA, Rennes Cedex, France

Jiří Srba
Aalborg University, Department of Computer Science, Denmark

Abstract We introduce a novel formalism of label-structured modal transition systems that combines the classical may/must modalities on transitions with structured labels that represent quantitative aspects of the model. On the one hand, the specification formalism is general enough to include models like weighted modal transition systems and allows the system developers to employ more complex label refinement than in the previously studied theories. On the other hand, the formalism maintains the desirable properties required by any specification theory supporting compositional reasoning. In particular, we study modal and thorough refinement, determinization, parallel composition, conjunction, quotient, and logical characterization of label-structured modal transition systems.

1. Introduction

Modern computing systems are often large and complex assemblies of numerous reactive and interacting components. The components are often designed by independent teams, working under a common agreement what the interface of each component should be. Consequently, the search for mathematical foundations which support *compositional reasoning* about interfaces is a major research goal. The framework should support inferring properties of the global implementation, and designing and advisedly reusing components.

In a logical setting, interfaces are specifications and components that implement an interface are understood as models/implementations. Specification theories should support various features including (1) *refinement*, which allows to compare specifications as well as to replace a specification by another one in a larger design, (2) *structural composition*, which allows to combine specifications of different components, (3) *logical conjunction*, expressing the intersection of the set of requirements expressed by two or more specifications, and last but not least, (4) a *quotient operator* that, given two specifications S and T, synthesizes the largest (w.r.t. refinement) specification that can be composed with S in order to refine T.

For sequential systems the classical notion of Denotational Semantics, founded by Scott and Strachey, provides a rich mathematical foundation for successfully describing the semantics of many sequential programming languages and systems [Gor79, Sto77] where components, i.e. programs, are basically modelled as computable functions from the domain of input values to the domain of output values. Most importantly, the semantics of a composite program is expressed in terms of the semantics of its components thus supporting compositional reasoning. A similar well-established specification theory for sequential systems is that of Hoare Logic [Hoa69], where a program is specified by pairs of pre- and post-conditions on states. In particular, Hoare Logic comes equipped with all the ingredients required and described above for a specification theory, with *"strongest postcondition"* and *"weakest precondition"* transformers providing the means for composing and quotienting specifications with respect to sequential composition.

Process algebras such as CCS [Mil80] and CSP [Hoa85] provide a corresponding mathematical foundation for concurrent and reactive systems. Here systems are semantically understood as labelled transition systems [Plo81] describing their interaction capabilities and dynamic evolution. Based on the labelled transition system semantics, several *equivalences* and *preorders* have been proposed [vG90] in order to capture different aspects of the extensional behaviour of a process. This results in specification theories where both the specification and the implementation are expressed within the same formalism, e.g. CCS, and with a preferred preorder or equivalence determining the *satisfaction* of an implementation with respect to a specification. To achieve the goal of compositional analysis great care has normally been taken to ensure that the preorders and equivalences are substitutive with respect to the various process constructions, e.g. parallel composition, including the notions of *observational equivalence* [Mil80] and *bisimulation equivalence* [Par81, Mil83] used in CCS.

The specification theory of *modal transition systems* [LT88a] grew out of a series of

attempts to achieve a more flexible and easy-to-use compositional development methodology for CCS. For the initial motivation consider the so-called *stepwise refinement method* to be carried out in CCS. A specification (in CCS) S of some desired system is given. The task is to find an implementation I of S such that $I \equiv S$, where $\equiv$ may be observational (or bisimulation) equivalence. In a first refinement-step S might be refined to a composite specification of the form $C[S_1, S_2]$, where the context C is some CCS-construct (e.g. parallel composition) and S_1 and S_2 are subspecifications. Now we may have different teams working independently towards implementations I_1 and I_2 of the subspecifications S_1 and S_2. Given the congruence property of observational equivalence, it will now suffice to establish the equivalences below to conclude, in a compositional manner, that the assembled implementation $C[I_1, I_2]$ satisfies the original specification S:

$$C[S_1, S_2] \equiv S \tag{1}$$
$$I_1 \equiv S_1 \tag{2}$$
$$I_2 \equiv S_2 \tag{3}$$

However, looking more carefully at the stepwise refinement above, we notice that (2) and (3) require S_i and I_i $(i = 1, 2)$ to be proved congruent, i.e. interchangeable, in *any* context and not just interchangeable in the context of C in which they are actually going to be placed. We are therefore asked to prove more than what seems necessary. Moreover, the subspecifications S_i $(i = 1, 2)$ may have to specify behaviours that are not at all relevant in the context C. Again it seems that the above compositional analysis can be substantially harder than necessary.

In order to reduce the work of (1-3), the notion of *context-dependent* or *relativized* bisimulation was introduced in [Lar85, Lar87]. Here, in order to reduce the overall effort, the observational equivalence $\equiv$ is relativized with information about the context C. The required proofs $I_i \equiv S_i$ can thus be replaced with proofs of the more specific $I_i \equiv_e S_i$ where e is some partial information (stated as a labelled transition system) about the context C. The work [LM87, LM92] applies the relativized bisimulation to the compositional verification of the Alternating Bit Protocol, and [PS00] introduces a proof technique for polymorphic Pi-Calculus based on polymorphic types which can be seen as a "disciplined instance" of relativized bisimulation. A more recent usage of relativized bisimulation includes the use of environment information to produce environment-specific (reduced) code from embedded system specifications [LLW05] and to generate relevant test sequences from real-time specifications [LMN05].

The introduction of modal transition system was pre-dated by the simpler formalisms of *partial specifications* [LT88b] and the corresponding notion of partial bisimulation. Roughly speaking, partial specifications are labelled transition systems with certain (specification) states being interpreted as completely unspecified. As such, partial specifications are very similar to that of processes with divergence and the so-called pre-bisimulation [Sti87, SW89]. However, though allowing for simple and intuitive subspecifications on several examples, the specification theory constituted by partial specifications is closed under neither conjunction nor quotienting.

Compared with partial specifications, the introduction of modal transition systems [LT88a] resulted in a specification theory much closer to logic (see [BL92] for a logical characterization of the expressive power of modal transition systems), thus still with a behavioural semantics allowing for easy composition with respect to process constructions. In short, modal transition systems are labelled transition systems equipped with two types of transitions: *must* transitions that are mandatory for any implementation, and *may* transitions which are optional for an implementation. Refinement of modal transition systems now essentially consists in iteratively resolving the unsettled status of may transitions: either by removing them or by turning them into must transitions.

It is well admitted that modal transition systems and their extensions (e.g., [Rac08]) match all the requirements of a good specification theory. There is also no doubt that the formalism is expressive enough to encode complex industrial problems (see e.g., [COM11, SPE10]). Moreover, the model has applications in other contexts, which include the verification of product lines [FUB06, GLS08a, LNW07b] and a counterexample-guided abstraction refinement technique for transition systems [GHJ01].

While searching for a specification theory for embedded systems, it is not only the functional requirements [LNW07b, FP07, BMSH10] of system behaviours that are of importance. The theory should be also capable of expressing constraints for several non-functional properties such as timing, energy-consumption, band-width etc. Recently such efforts have been of high interest in the theory community [CDL+10, KKN09, DKL+11, BLPR09, BPR09, DLL+10].

For different non-functional extensions it is common that similar proof techniques are used to argue about the specification formalisms. In this article, we present a specification theory that unifies several of the proof techniques described in the literature by introducing a general framework of *label-structured modal transition system*. Specializations of the framework include, apart from the well-known instances like unlabelled/labelled modal transition systems, also a new specification theory for weighted and multi-weighted transition systems that were studied only recently [JLS12]. Other formalisms like timed modal transitions systems can be embedded into the framework as argued in [BLPR09] where the authors show that operations defined on some classes of timed modal specifications can be reduced to questions on modal transition systems by using a region-based abstraction.

In this article, we study the classical questions related to the formalism of label-structured modal transition systems: (i) modal and thorough refinement, consistency and pruning, determinism and deterministic hull (in Section 2), (ii) parallel composition, conjunction and quotient (in Section 3) and (iii) logical characterization including generalized model checking (in Section 4).

As a result, we offer in a self-contained manner a full specification theory of label-structured modal transition systems. The theory specializes to some well-known formalisms studied earlier but at the same time also provides novel results for instances such as weighted and multi-weighted modal transition systems.

2. Label-Structured Modal Transition Systems

We shall now introduce the notion of label-structured modal transition systems and some basic properties of the formalism. Before that we need to define the notion of labels and label-sets used during the system design and specification refinement.

Definition 1 (Label-set)**.** A *label-set* is a partially ordered set of labels $(K, \sqsubseteq)$ such that $\bot \in K$ (modelling inconsistency) is the least element of K.

A label $k \in K \setminus \{\bot\}$ is called an *implementation label* if $k' \sqsubseteq k$ implies $k' = k$ for all $k' \in K \setminus \{\bot\}$. In other words, implementation labels are all elements in K just above $\bot$. The set of all implementation labels of $(K, \sqsubseteq)$ is denoted by $\mathrm{Imp}(K, \sqsubseteq)$. To each label $k \in K$ we associate the set $[\![k]\!]$ of all implementation labels below k by

$$[\![k]\!] = \{k' \in \mathrm{Imp}(K, \sqsubseteq) \mid k' \sqsubseteq k\}.$$

Definition 2 (Well-formed label-set)**.** A label-set $(K, \sqsubseteq)$ with the least element $\bot \in K$ is called *well-formed* if $[\![k]\!] \neq \emptyset$ for every $k \in K \setminus \{\bot\}$.

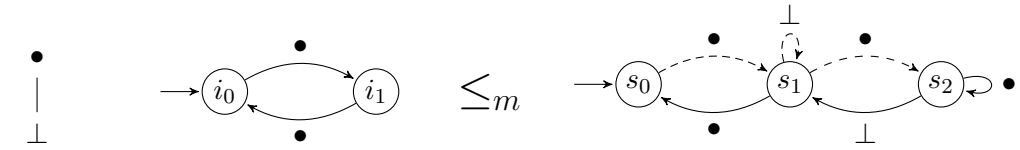
Well-formedness of label-sets ensures consistency of the label refinement relation $\sqsubseteq$, in other words it should always be possible to refine any label into an implementation label. We can now define label-structured modal transition systems that combine the underlying may/must transition relation known from modal transition systems with the label structure defined above.

Definition 3 (Label-structured modal transition system)**.** A *label-structured modal transition system (LSMTS)* is a tuple $(S, s_0, (K, \sqsubseteq), \dashrightarrow, \longrightarrow)$ where S is a set of states with the initial state $s_0 \in S$, $(K, \sqsubseteq)$ is a well-formed label-set, $\dashrightarrow \subseteq S \times K \times S$ is the *may* transition relation, and $\longrightarrow \subseteq S \times K \times S$ is the *must* transition relation such that $\longrightarrow \subseteq \dashrightarrow$.

We write $s \overset{k}{\dashrightarrow} s'$ if $(s, k, s') \in \dashrightarrow$. If for some $k \in K$ no state $s' \in S$ exists such that $s \overset{k}{\dashrightarrow} s'$ we write $s \not\overset{k}{\dashrightarrow}$, and if there exists some $s' \in S$ such that $s \overset{k}{\dashrightarrow} s'$ we write $s \overset{k}{\dashrightarrow}$. The aforementioned notations apply also to $\longrightarrow$. By abuse of notation, we use S to denote an LSMTS $(S, s_0, (K_S, \sqsubseteq_S), \dashrightarrow_S, \longrightarrow_S)$ and the subscripts are omitted if they are clear from the context. The notation (s, S) denotes the LSMTS S with the initial state s_0 replaced by s. The class of all LSMTSs with the well-formed label-set $(K, \sqsubseteq)$ is denoted by $\mathcal{M}_{(K,\sqsubseteq)}$, and we typically use capital letters S, T, U to range over this class.

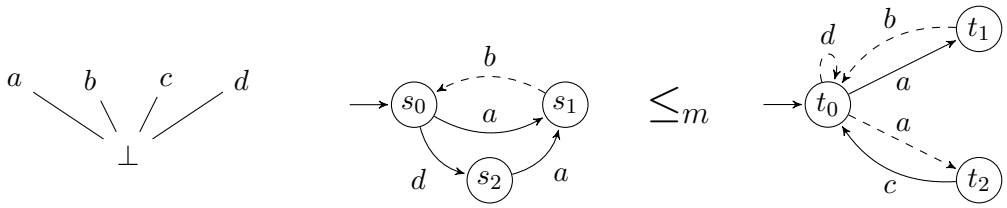
An LSMTS S is called an *implementation* if $\longrightarrow = \dashrightarrow$ and all labels on the transitions are implementation labels, that is, for all $s \overset{k}{\longrightarrow} s'$ in S we have $k \in \mathrm{Imp}(K, \sqsubseteq)$. The class of all implementations with well-formed label-set $(K, \sqsubseteq)$ is denoted by $\mathcal{I}_{(K,\sqsubseteq)}$, and we typically use capital letters I and J to range over this class.

In the following, LSMTSs will be often represented as graphs with the convention that whenever two states are connected by both a must and a may transition under the same label, then we draw only the must transition.



(a) $\sqsubseteq$ (b) I (c) S

Figure 1.: Unlabelled modal transition system over the label-set $K_{unlabelled}$



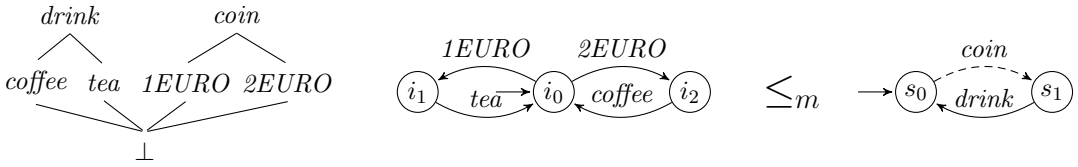
(a) $\sqsubseteq$ (b) S (c) T

Figure 2.: Modal transition systems over the label-set K_{action}

Example 4. The most trivial instance of LSMTSs is obtained by choosing the well-formed label-set $K_{unlabelled} = (\{\bot, \bullet\}, \sqsubseteq)$ where $\sqsubseteq = \{(\bot, \bot), (\bot, \bullet), (\bullet, \bullet)\}$ illustrated in Figure 1(a). This label-set gives rise to unlabelled modal transition systems where $\bullet$ models a single implementation label and $\bot$ is the inconsistency label. An example is shown in Figure 1(b) and (c). The LSMTS I in Figure 1(b) is an implementation because every label is an implementation label and the may and must transition relations coincide. Note that the LSMTS in Figure 1(c) is not an implementation as (i) there are transitions labelled with $\bot$ and (ii) there are several may transitions without the corresponding must ones. The definition and explanation of modal refinement, denoted $\leq_m$, is deferred to Section 2.2. □

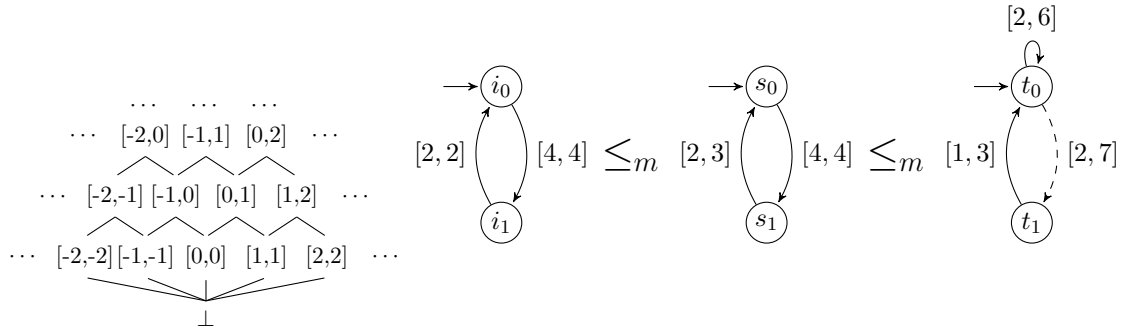
Example 5. A well-known instance of the framework is obtained by considering a finite set of actions Σ and defining a well-formed label-set K_{action} by $K_{action} = (\Sigma \cup \{\bot\}, \sqsubseteq)$ where $a \sqsubseteq b$ if and only if $a = \bot$ or $a = b$. Here all labels (apart from $\bot$) are implementation labels and this setting corresponds exactly to the class of modal transition systems [LT88a]. Illustration of the label-set K_{action} and two examples of modal transition systems are given in Figure 2. □

Example 6. As another example of a well-formed label-set demonstrating a more interesting label refinement, we can consider the following structure $K_{machine} = (\{drink, coffee, tea, coin, 1EURO, 2EURO, \bot\}, \sqsubseteq)$ where the ordering $\sqsubseteq$ is given in Figure 3(a). Here it is possible to provide a high-level specification of a vending machine by using the labels *drink* and *coin* that can be later in a concrete implementation refined into the implementation labels *coffee* and *tea*, and *1EURO* and *2EURO*, respectively. □



(a) $\sqsubseteq$ (b) I (c) S

Figure 3.: Vending machines over the label-set $K_{machine}$



(a) $\sqsubseteq$ (b) I (c) S (d) T

Figure 4.: Weighted modal automata over $K_{weighted}$

Example 7. Another instance of the framework is called weighted modal automata. Here the well-formed label-set $K_{weighted}$ is given as a set of integer intervals with the natural inclusion ordering, formally $K_{weighted} = (K, \sqsubseteq)$ where $K = \{[a, b] \mid a, b \in \mathbb{Z} \text{ s.t. } a \leq b\} \cup \{\bot\}$ and $[a', b'] \sqsubseteq [a, b]$ if $a \leq a'$ and $b' \leq b$, $\bot \sqsubseteq [a, b]$, and $\bot \sqsubseteq \bot$, for all $a, a', b, b' \in \mathbb{Z}$. It follows that implementation labels are singleton sets of the form $[a, a]$ where $a \in \mathbb{Z}$. Consult Figure 4 for the illustration of $K_{weighted}$ and for three examples of weighted modal automata. The automaton I is an implementation while S and T are not. □

2.1. Product of Labels

In this subsection we will discuss a product construction on labels which will allow us to form (by a general construction) new instances of the framework from existing ones.

Definition 8 (Product)**.** Let $(K_1, \sqsubseteq_1)$ and $(K_2, \sqsubseteq_2)$ be two label-sets with the least elements $\bot_1$ and $\bot_2$, respectively. The *product* $(K_1, \sqsubseteq_1) \otimes (K_2, \sqsubseteq_2)$ of the two label-sets is a label-set $(K, \sqsubseteq)$ where $K = \big((K_1 \setminus \{\bot_1\}) \times (K_2 \setminus \{\bot_2\})\big) \cup \{\bot\}$ and $(k_1', k_2') \sqsubseteq (k_1, k_2)$ if $k_1' \sqsubseteq_1 k_1$ and $k_2' \sqsubseteq_2 k_2$ for all $k_1, k_1' \in K_1 \setminus \{\bot_1\}$ and all $k_2, k_2' \in K_2 \setminus \{\bot_2\}$, and $\bot \sqsubseteq \ell$ for all $\ell \in K$.

It is easy to observe that the product construction preserves well-formedness and implementations are derived component-wise as stated in the following lemma.

Lemma 9. *Let $(K_1, \sqsubseteq_1)$ and $(K_2, \sqsubseteq_2)$ be well-formed label-sets. Then*

1. $(K_1, \sqsubseteq_1) \otimes (K_2, \sqsubseteq_2)$ *is a well-formed label-set, and*
2. $\mathrm{Imp}((K_1, \sqsubseteq_1) \otimes (K_2, \sqsubseteq_2)) = \mathrm{Imp}(K_1, \sqsubseteq_1) \times \mathrm{Imp}(K_2, \sqsubseteq_2)$.

Using the product construction of label-sets, we can e.g. combine the previously introduced well-formed label-sets K_{action} and $K_{weighted}$ from Examples 5 and 7 into weighted modal transition systems using the label-set $K_{action} \otimes K_{weighted}$ or into multi-weighted modal transition systems using the label-set $K_{action} \otimes K_{weighted} \otimes K_{weighted} \otimes \ldots \otimes K_{weighted}$ and further combine these with other quantitative aspects.

2.2. Refinement

We shall now define the notion of modal refinement that combines the label refinement, given by the partial ordering on the label-set, with the allowed transitions that may be present and required transitions that must be present. It is a generalization of the original notion of modal refinement over classical modal transition systems [LT88a].

Definition 10 (Modal refinement)**.** Let $S, T \in \mathcal{M}_{(K,\sqsubseteq)}$ be two LSMTSs with initial states s_0 and t_0, respectively. We say that S *modally refines* T, written $S \leq_m T$, if there exists a relation $R \subseteq S \times T$ with $(s_0, t_0) \in R$ such that for every $(s, t) \in R$:

1. whenever $s \overset{k}{\dashrightarrow} s'$ then there is $t \overset{\ell}{\dashrightarrow} t'$ such that $k \sqsubseteq \ell$ and $(s', t') \in R$, and
2. whenever $t \overset{\ell}{\longrightarrow} t'$ then there is $s \overset{k}{\longrightarrow} s'$ such that $k \sqsubseteq \ell$ and $(s', t') \in R$.

The *implementation semantics* of an LSMTS $S \in \mathcal{M}_{(K,\sqsubseteq)}$ is defined as the class $[\![S]\!]$ of all implementations refining S, i.e. $[\![S]\!] = \{I \in \mathcal{I}_{(K,\sqsubseteq)} \mid I \leq_m S\}$.

Example 11. Refinement of modal transition systems labelled with actions (see Example 5) is illustrated in Figure 2. The system S is a modal refinement of the system T, and the relation demonstrating this is given by $\{(s_0, t_0), (s_1, t_1), (s_2, t_0)\}$. Note that S is not an implementation yet, as it contains a may transition under b without a must transition under the same label. □

Example 12. Consider the label-set $K_{machine}$ from Example 6. A specification of a vending machine is depicted in Figure 3(c). It allows to enter a coin and, should this happen, it requires that a drink is returned to the customer. One of the possible implementations (where all labels are implementation labels and the may and must transition relations coincide) of this specification is given in Figure 3(b). The modal refinement between the implementation and specification is easily demonstrated by the relation $\{(i_0, s_0), (i_1, s_1), (i_2, s_1)\}$. □

Example 13. Refinement of weighted modal automata (see Example 7) is illustrated in Figure 4. The relation $\{(s_0, t_0), (s_1, t_0)\}$ is witnessing the modal refinement between Figure 4d and 4c. Note that the refined specification in Figure 4c is not an implementation yet as it contains the label $[2, 3]$ which is not an implementation label. We can thus refine it further, ending up with an implementation as seen in Figure 4b. The witnessing relation is $\{(i_0, s_0), (i_1, s_1)\}$. □

Lemma 14. *The modal refinement relation $\leq_m$ is a preorder.*

Proof. Reflexivity is trivial. Transitivity can be seen as follows. Let $S, T, U \in \mathcal{M}_{(K,\sqsubseteq)}$ be three LSMTSs with their initial states s_0, t_0, u_0 such that $S \leq_m T \leq_m U$. From the assumption $S \leq_m T$ we know that there exists a witnessing relation $R_1 \subseteq S \times T$, and from the assumption $T \leq_m U$ we know that there exists a witnessing relation $R_2 \subseteq T \times U$. We define a relation $R \subseteq S \times U$ by the relational composition of R_1 and R_2, i.e.

$$R = \{(s, u) \mid \exists t \in T : (s, t) \in R_1 \text{ and } (t, u) \in R_2\}.$$

We show that R is proving $S \leq_m U$. Obviously $(s_0, u_0) \in R$. Now, let $(s, u) \in R$ be an arbitrary element of R. Let $t \in T$ be a state such that $(s, t) \in R_1$ and $(t, u) \in R_2$.

1. Assume $s \overset{k_s}{\dashrightarrow} s'$. From $(s, t) \in R_1$ it follows that there exists $t \overset{k_t}{\dashrightarrow} t'$ such that $k_s \sqsubseteq k_t$ and $(s', t') \in R_1$. Then, as $(t, u) \in R_2$, we get $u \overset{k_u}{\dashrightarrow} u'$ such that $k_t \sqsubseteq k_u$ and $(t', u') \in R_2$, hence $(s', u') \in R$ and by transitivity of $\sqsubseteq$ also $k_s \sqsubseteq k_u$.

2. Symmetric to the previous direction.

■

Modal refinement induces an equivalence relation on LSMTS. We say that S and T are *equivalent*, denoted by $S \equiv_m T$, if both $S \leq_m T$ and $T \leq_m S$ are satisfied.

Lemma 15. *Let $I, J \in \mathcal{I}_{(K,\sqsubseteq)}$ be two implementations. Then $I \leq_m J$ implies $I \equiv_m J$.*

Proof. Given a relation R witnessing $I \leq_m J$ we can use $R^{-1} = \{(j, i) \mid (i, j) \in R\}$ to prove that $J \leq_m I$. Let $(j, i) \in R^{-1}$.

1. Assume that $i \xrightarrow{k} i'$. Remember that then also $i \overset{k}{\dashrightarrow} i'$. From the fact that $(i, j) \in R$ it follows that there exists $j \overset{\ell}{\dashrightarrow} j'$ such that $k \sqsubseteq \ell$ and $(i', j') \in R$. Since J is an implementation, we get that $k = \ell$ and $j \xrightarrow{\ell} j'$. Obviously, $(j', i') \in R^{-1}$.

2. The other direction is symmetric.

■

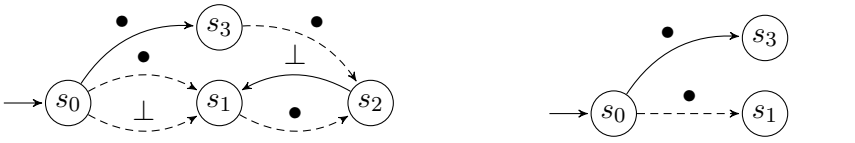
The notion of modal refinement can be understood as refinement defined at the syntactical level as it directly relates the states of two specifications. A semantically motivated notion of refinement, usually called *thorough refinement*, says that S is a refinement of T if every implementation of S is also an implementation of T.

Definition 16 (Thorough refinement)**.** Let $S, T \in \mathcal{M}_{(K,\sqsubseteq)}$ be two LSMTSs. We say that S *thoroughly refines* T, written $S \leq_t T$, if $[\![S]\!] \subseteq [\![T]\!]$.

It is an expected result that modal refinement implies thorough refinement, as stated in the following soundness theorem. The opposite implication does not hold in general and details are discussed in Section 2.4.

Theorem 17 (Soundness)**.** *Let $S, T \in \mathcal{M}_{(K,\sqsubseteq)}$ be two LSMTSs. Then $S \leq_m T$ implies $S \leq_t T$.*

Proof. Follows immediately from the transitivity of modal refinement (see Lemma 14). ■



(a) A consistent, but locally inconsistent LSMTS S

(b) The locally consistent result $\rho(S)$ of pruning S

Figure 5.: Example of the pruning operator

2.3. Consistency and Pruning

Similar to the classical notion of consistency, an LSMTS S is consistent if it has at least one implementation.

Definition 18 (Consistency)**.** Let $S \in \mathcal{M}_{(K,\sqsubseteq)}$. The LSMTS S is *consistent* if $[\![S]\!] \neq \emptyset$.

Consistency is a semantical notion and in the rest of this article it will be useful to introduce also a syntactical notion of consistency, called local consistency.

Definition 19 (Local consistency)**.** Let $S \in \mathcal{M}_{(K,\sqsubseteq)}$. A state $s \in S$ is *locally consistent* if $s \not\xrightarrow{\bot}$. The LSMTS S is *locally consistent* if all states of S are locally consistent.

From our assumption of well-formedness of label-sets, it follows that any locally consistent S has at least one implementation, thus local consistency implies consistency. The converse is not true as explained in the following example.

Example 20. Consider the LSMTS S presented in Figure 5(a) with the label-set $K_{unlabelled}$ from Example 4. The system S is clearly not locally consistent but it is consistent as an implementation with just two states connected by a must (and may) transition labelled with $\bullet$ is an implementation of S. □

We shall now define a pruning operator that removes locally inconsistent states.

Definition 21 (Pruning)**.** Let $S \in \mathcal{M}_{(K,\sqsubseteq)}$ be an LSMTS and let $B \subseteq S$ be a subset of its states. Let

$$pre(B) = \{s \in S \mid s \xrightarrow{k} s' \text{ and } s' \in B \text{ for some } k \in K\}$$

and $pre^0(B) = B$, $pre^{j+1}(B) = pre(pre^j(B))$ for $j \geq 0$, and $pre^*(B) = \bigcup_{j\geq 0} pre^j(B)$. The *pruning* $\rho(S)$ of S is defined if $s_0 \notin pre^*(\mathsf{bad})$ where $\mathsf{bad} = \{s \in S \mid s \xrightarrow{\bot}\}$, and in this case, $\rho(S)$ is the LSMTS $(S^\rho, s_0, (K, \sqsubseteq), \dashrightarrow_\rho, \longrightarrow_\rho)$ where

$$\begin{aligned} S^\rho &= S \setminus pre^*(\mathsf{bad}), \\ \dashrightarrow_\rho &= \dashrightarrow \cap \left(S^\rho \times (K \setminus \{\bot\}) \times S^\rho\right), \text{ and} \\ \longrightarrow_\rho &= \longrightarrow \cap \left(S^\rho \times (K \setminus \{\bot\}) \times S^\rho\right). \end{aligned}$$

It is clear that for an LSMTS with n states one can compute $pre^*(\mathsf{bad})$ by finitely many iterations, more precisely $pre^*(\mathsf{bad}) = pre^n(\mathsf{bad})$ in this case. Also note that well-definedness of pruning is equivalent to the absence of a path of must transitions from the initial state to a locally inconsistent state (that enforces inconsistency via must transition labelled with $\bot$).

Figure 5 shows the application of the pruning operator ρ to the system S and one can easily observe that $\rho(S) \leq_m S$. Pruning also does not remove any implementation. A summary of the properties of pruning is given in the following proposition.

Proposition 22. *Let $S \in \mathcal{M}_{(K,\sqsubseteq)}$. If $\rho(S)$ is defined, then*

1. *$\rho(S)$ is locally consistent,*
2. *$\rho(S) \leq_m S$,*
3. *$[\![\rho(S)]\!] = [\![S]\!]$, and*
4. *for any locally consistent $T \in \mathcal{M}_{(K,\sqsubseteq)}$, if $T \leq_m S$ then $T \leq_m \rho(S)$.*

Moreover, $\rho(S)$ is defined if and only if S is consistent.

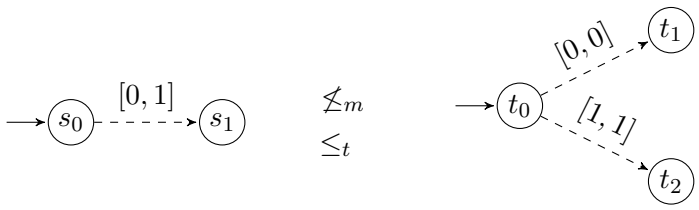
Proof. 1. As all labels $\bot$ were removed in $\rho(S)$, it is trivially locally consistent.

2. We will show that the relation $R = \{(s, s) \mid s \in S \setminus pre^*(\mathsf{bad})\}$ is a refinement relation in order to argue that $\rho(S) \leq_m S$. Let $(s, s) \in R$. If $s \overset{k}{\dashrightarrow}_\rho s'$ in $\rho(S)$ then this by the construction implies that $s' \in S \setminus pre^*(\mathsf{bad})$. Clearly we have also $s \overset{k}{\dashrightarrow} s'$ in S and $(s', s') \in R$. On the other hand, if $s \overset{k}{\longrightarrow} s'$ in S then $s' \notin pre^*(\mathsf{bad})$ as $s \notin pre^*(\mathsf{bad})$, which means that $s \overset{k}{\longrightarrow}_\rho s'$ also in $\rho(S)$ and $(s', s') \in R$.

3. The inclusion $[\![\rho(S)]\!] \subseteq [\![S]\!]$ follows from the fact that $\rho(S) \leq_m S$. Let $I \in [\![S]\!]$. This means that there is a refinement relation R demonstrating that $I \leq_m S$. We will argue that R is also a refinement relation demonstrating $I \leq_m \rho(S)$. However, this easily follows from the observation that R cannot contain any state from $pre^*(\mathsf{bad})$, because otherwise a must path to the label $\bot$ will be enforced in I too, but then I is not an implementation.

4. The same argumentation as in the previous point applies also here. Any refinement relation demonstrating $T \leq_m S$ can be used to establish also $T \leq_m \rho(S)$. In order to apply the reasoning as above, it is important that T does not contain any transition with the label $\bot$.

For the last claim, observe that if $\rho(S)$ is not defined then there is a must path from its initial state to a state requiring a transition under $\bot$. Any implementation then has to contain such a path to a state requiring a transition under $\bot$, but then it is not an implementation. On the other hand, if $\rho(S)$ is defined then we can change every transition in $\rho(S)$ to a must transition and replace every label by some implementation label below it (possible thanks to well-formedness of the label-set) in order to construct an implementation of $\rho(S)$, which is also an implementation of S as $\rho(S) \leq_m S$. ■



(a) S (b) T

Figure 6.: Incompleteness of modal refinement demonstrated by systems S and T

2.4. Determinism and Completeness of Refinement

In general, thorough refinement does not imply modal refinement. A counterexample, using the label-set $K_{weighted}$ (see Example 13), is given in Figure 6. Clearly, the transition $s_0 \overset{[0,1]}{\dashrightarrow} s_1$ cannot be matched by any of the two transitions from t_0 as their labels are less general than $[0,1]$. Hence $S \not\leq_m T$. On the other hand, any implementation of S is either empty or it is a tree of height one with the outgoing edges labelled by either $[0,0]$ or $[1,1]$. All such implementations are also refinements of the system T.

It is known that for classical modal transition systems thorough refinement implies the modal one, under the assumption of determinism [BKLS09b]. We can generalize this result to the class of label-structured modal transition systems. Before we define when an LSMTS is deterministic, we first define when two labels k_1, k_2 are unifiable, that is, if there is another label k which overlaps with k_1 and k_2 with respect to their sets of implementation labels.

Definition 23 (Unifiable labels)**.** Two labels $k_1, k_2 \in K$ are called *unifiable* if there exists $k \in K$ such that $[\![k]\!] \cap [\![k_1]\!] \neq \emptyset$ and $[\![k]\!] \cap [\![k_2]\!] \neq \emptyset$.

Then, determinism expresses that for any two outgoing may transitions from the same state under two different labels k_1 and k_2, the labels k_1 and k_2 are not unifiable.

Definition 24 (Determinism)**.** A LSMTS S is called *deterministic* if for any state $s \in S$ and any two transitions $s \overset{k_1}{\dashrightarrow} s_1'$ and $s \overset{k_2}{\dashrightarrow} s_2'$, if k_1 and k_2 are unifiable, then $k_1 = k_2$ and $s_1' = s_2'$.

Returning to Figure 6 we can realize that the system T is not deterministic as there is a branching of the transitions with labels $[0,0]$ and $[1,1]$, while there exists a label $[0,1]$ such that $[\![[0,1]]\!] \cap [\![[0,0]]\!] \neq \emptyset$ and $[\![[0,1]]\!] \cap [\![[1,1]]\!] \neq \emptyset$.

A very natural assumption that has to be imposed on the label-sets later on in order to show completeness of modal refinement, is completeness of label refinement: inclusion of implementation labels implies label refinement.

Definition 25 (Completeness of label refinement)**.** Let $(K, \sqsubseteq)$ be a label-set. Label refinement $\sqsubseteq$ is *complete* if for all $k, \ell \in K$, $[\![k]\!] \subseteq [\![\ell]\!]$ implies $k \sqsubseteq \ell$.

All examples of label-sets provided in this article satisfy this property. Note that label refinement is always sound by definition, i.e. $k \sqsubseteq \ell$ implies $[\![k]\!] \subseteq [\![\ell]\!]$ by transitivity of label refinement.

Under the assumption of (1) completeness of label refinement, (2) determinism of the refined LSMTS, and (3) local consistency of the refining LSMTS, thorough refinement implies the modal one.

Theorem 26 (Completeness)**.** *Let $(K, \sqsubseteq)$ be a well-formed label-set for which label refinement $\sqsubseteq$ is complete. Let $S, T \in \mathcal{M}_{(K,\sqsubseteq)}$ with initial states s_0 and t_0, respectively, such that S is locally consistent and T is deterministic. Then $S \leq_t T$ implies $S \leq_m T$.*

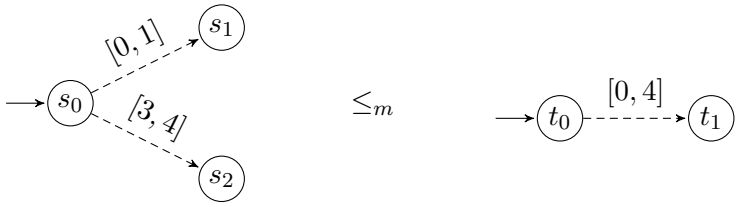
Proof. Assume that $S \leq_t T$. We define a relation $R \subseteq S \times T$ as the smallest relation satisfying:

1. $(s_0, t_0) \in R$,
2. if $(s,t) \in R$, $s \overset{k}{\dashrightarrow} s'$, $t \overset{\ell}{\dashrightarrow} t'$, and $[\![k]\!] \cap [\![\ell]\!] \neq \emptyset$ then $(s', t') \in R$.

First, we show a technical result (that we use later on) saying that any $(s,t) \in R$ satisfies $[\![(s,S)]\!] \subseteq [\![(t,T)]\!]$. For $(s_0, t_0) \in R$, we have $[\![(s_0, S)]\!] = [\![S]\!] \subseteq [\![T]\!] = [\![(t_0, T)]\!]$ from the assumption $S \leq_t T$. Now, let $(s,t) \in R$ such that $[\![(s,S)]\!] \subseteq [\![(t,T)]\!]$ and assume that $s \overset{k}{\dashrightarrow} s'$, $t \overset{\ell}{\dashrightarrow} t'$, and $[\![k]\!] \cap [\![\ell]\!] \neq \emptyset$. Let $I' \in [\![(s', S)]\!]$ and let $m \in [\![k]\!] \cap [\![\ell]\!]$ which exists by the construction. Then, since S is locally consistent, there exists an implementation $(i_0, I) \in [\![(s,S)]\!]$ such that $i_0 \xrightarrow{m} i'$ and $(i', I) \leq_m I'$. From $[\![(s,S)]\!] \subseteq [\![(t,T)]\!]$ it follows that $I \in [\![(t,T)]\!]$. Then there exists a transition $t \overset{\ell'}{\dashrightarrow} t''$ such that $(i', I) \in [\![(t'', T)]\!]$ and $m \in [\![\ell']\!]$. Now, we have $t \overset{\ell}{\dashrightarrow} t'$ and $t \overset{\ell'}{\dashrightarrow} t''$ such that $m \in [\![\ell]\!] \cap [\![\ell']\!]$, hence ℓ and ℓ' are unifiable. As T is deterministic it follows that $\ell = \ell'$ and $t' = t''$, so $(i', I) \in [\![(t', T)]\!]$. Finally, from $(i', I) \leq_m I'$ and Lemma 15 it follows that $(i', I) \equiv_m I'$ and hence $I' \in [\![(t', T)]\!]$.

Now we show that R is a relation witnessing $S \leq_m T$. Clearly $(s_0, t_0) \in R$. Let $(s,t) \in R$.

1. Assume $s \overset{k}{\dashrightarrow} s'$. By local consistency of S we can assume that $k \neq \bot$. Then, for each implementation label $m \in [\![k]\!]$, there exists an implementation $I_m \in [\![(s,S)]\!]$ such that $i_0 \xrightarrow{m} i'$. We also know that $I_m \in [\![(t,T)]\!]$ because $[\![(s,S)]\!] \subseteq [\![(t,T)]\!]$. Hence there exists a transition $t \overset{\ell_m}{\dashrightarrow} t'_m$ such that $m \in [\![\ell_m]\!]$. We have to show that, for all $m \in [\![k]\!]$, the labels ℓ_m are the same. Suppose that there are $m_1, m_2 \in [\![k]\!]$ and transitions $t \overset{\ell_{m_1}}{\dashrightarrow} t'_{m_1}$ and $t \overset{\ell_{m_2}}{\dashrightarrow} t'_{m_2}$ such that $m_1 \in [\![\ell_{m_1}]\!]$ and $m_2 \in [\![\ell_{m_2}]\!]$. Then, since $m_1 \in [\![\ell_{m_1}]\!] \cap [\![k]\!]$ and $m_2 \in [\![\ell_{m_2}]\!] \cap [\![k]\!]$ and T is deterministic, it follows that $\ell_{m_1} = \ell_{m_2}$ and $t'_{m_1} = t'_{m_2}$. It follows that there is a unique transition $t \overset{\ell}{\dashrightarrow} t'$ such that $m \in [\![\ell]\!]$ for all implementation labels $m \in [\![k]\!]$, this means $[\![k]\!] \subseteq [\![\ell]\!]$ which implies $k \sqsubseteq \ell$ by completeness of label refinement. Moreover, by the definition of R, we get $(s', t') \in R$.



(a) S (b) $\mathcal{D}(S)$

Figure 7.: Determinization

2. Assume $t \xrightarrow{\ell} t'$. If $\ell = \bot$ then there is a transition $s \xrightarrow{\bot} s'$ in S, contradicting local consistency of S; hence $\ell \neq \bot$. Then, for each implementation $I \in [\![(t,T)]\!]$ we have that there exists a transition $i_0 \xrightarrow{m} i'$ for some label $m \in [\![\ell]\!]$. We know that $[\![(s,S)]\!] \subseteq [\![(t,T)]\!]$, so every implementation $(j_0, J) \in [\![(s,S)]\!]$ has a transition $j_0 \xrightarrow{m} j'$. It follows that S must have a transition $s \xrightarrow{k} s'$ such that $m \in [\![k]\!]$. Suppose that $[\![k]\!] \not\subseteq [\![\ell]\!]$, then there would exist an implementation $(\bar{j}_0, \bar{J}) \in [\![(s,S)]\!]$ having $\bar{j}_0 \xrightarrow{n} \bar{j}'$ with an implementation label $n \in [\![k]\!]$ not belonging to $[\![\ell]\!]$, which means that there must exist another transition in T, say $t \overset{\ell'}{\dashrightarrow} t''$ such that $n \in [\![\ell']\!]$. But then we have $m \in [\![\ell]\!] \cap [\![k]\!]$ and $n \in [\![\ell']\!] \cap [\![k]\!]$ which contradicts determinism of T. Thus, we have $[\![k]\!] \subseteq [\![\ell]\!]$, which implies $k \sqsubseteq \ell$ by completeness of label refinement. Moreover, by definition of R, we get $(s', t') \in R$.

■

2.5. Determinization

It is a well-known fact that, for almost all existing specification theories, deciding thorough refinement involves more complex decision procedures than the ones that can be used to decide modal refinement (see e.g. [BKLS09a] dealing with the classical modal transition systems). In the previous subsection, we have seen that in case of deterministic systems modal and thorough refinements coincide and can be decided by efficient syntactical fixed-point based algorithms. In Section 3.1 it will become moreover evident that determinism plays an important role in establishing several soundness results.

It is thus worth studying a general procedure that, given a nondeterministic LSMTS S, computes its smallest deterministic over-approximation $\mathcal{D}(S)$, called deterministic hull.

Example 27. Consider the system in Figure 7(a) where S is a LSMTS with label-set $K_{weighted}$. It is nondeterministic since there is the label $[0,4]$ for which $[\![[0,4]]\!] \cap [\![[0,1]]\!] \neq \emptyset$ and $[\![[0,4]]\!] \cap [\![[3,4]]\!] \neq \emptyset$. The best we can do is to approximate S by a deterministic LSMTS in Figure 7(b). However, $\mathcal{D}(S)$ is an over-approximation as $[\![S]\!] \subsetneq [\![\mathcal{D}(S)]\!]$ witnessed by the implementation with a single must transition labelled with $[2,2]$.

In the following, we will propose an algorithm that computes, for a given LSMTS S, a deterministic LSMTS $\mathcal{D}(S)$ such that $\mathcal{D}(S)$ is an over-approximation of S with the

property that it is a minimal one with respect to modal refinement. The construction is a generalization of the algorithm presented in [BKLS09b] and dealing specifically with modal transition systems.

We impose the following assumption on the label-set $(K, \sqsubseteq)$ necessary for applying the determinization algorithm. For any set $L \subseteq K$ of pairwise unifiable labels we require the existence of the least upper bound $lub(L) \in K$ such that $\ell \sqsubseteq lub(L)$ for all $\ell \in L$ and whenever $\ell \sqsubseteq \ell'$ for all $\ell \in L$ then $lub(L) \sqsubseteq \ell'$.

Definition 28 (Deterministic hull)**.** Let $S = (S, s_0, (K, \sqsubseteq), \dashrightarrow, \longrightarrow)$ be an LSMTS. The *deterministic hull of* S is defined by the LSMTS

$$\mathcal{D}(S) = (\mathcal{P}(S) \setminus \{\emptyset\}, \{s_0\}, (K, \sqsubseteq), \dashrightarrow_{\mathrm{D}}, \longrightarrow_{\mathrm{D}})$$

where the transition relations $\dashrightarrow_{\mathrm{D}}$ and $\longrightarrow_{\mathrm{D}}$ are defined as follows. Let $\mathcal{T} \in (\mathcal{P}(S) \setminus \{\emptyset\})$ be a state in $\mathcal{D}(S)$. For every maximal, nonempty set $L \subseteq \{k \mid s \overset{k}{\dashrightarrow},\ s \in \mathcal{T}\}$ of pairwise unifiable labels we have $\mathcal{T} \overset{\ell}{\dashrightarrow}_{\mathrm{D}} \mathcal{T}_\ell$ where $\ell = lub(L)$ and $\mathcal{T}_\ell = \{s' \in S \mid s \overset{k}{\dashrightarrow} s',\ s \in \mathcal{T},\ k \in L\}$. If, moreover, for each $s \in \mathcal{T}$ we have $s \overset{k}{\longrightarrow} s'$ for some $s' \in \mathcal{T}_\ell$ and some $k \in K$ such that $k \sqsubseteq \ell$, then $\mathcal{T} \overset{\ell}{\longrightarrow}_{\mathrm{D}} \mathcal{T}_\ell$.

Now we show that $\mathcal{D}(S)$ is the smallest deterministic over-approximation of S.

Theorem 29 (Soundness and minimality of determinization)**.** *Let $S \in \mathcal{M}_{(K,\sqsubseteq)}$. Then the following holds:*

1. *$\mathcal{D}(S)$ is deterministic,*
2. *$S \leq_m \mathcal{D}(S)$, and*
3. *for every deterministic $D \in \mathcal{M}_{(K,\sqsubseteq)}$, if $S \leq_m D$ then $\mathcal{D}(S) \leq_m D$.*

Proof.

1. Let $\mathcal{T}$ be a state in $\mathcal{D}(S)$ and assume that there exist two different transitions $\mathcal{T} \overset{\ell_1}{\dashrightarrow} \mathcal{T}_{\ell_1}$ and $\mathcal{T} \overset{\ell_2}{\dashrightarrow} \mathcal{T}_{\ell_2}$ such that ℓ_1 and ℓ_2 are unifiable. It follows that there exists a least upper bound $\ell \in K$ of ℓ_1 and ℓ_2. Hence all labels below ℓ_1 are unifiable with all labels below ℓ_2. This contradicts the fact that ℓ_1 and ℓ_2 are the least upper bounds of maximal, unifiable sets of labels.

2. We define a relation $R \subseteq S \times \mathcal{D}(S)$ by $R = \{(s, \mathcal{T}) \mid s \in \mathcal{T}\}$. Clearly, $(s_0, \{s_0\}) \in R$ for the respective initial states. Now, let $(s, \mathcal{T}) \in R$.

   First, assume that $s \overset{k}{\dashrightarrow} s'$. Then, since $s \in \mathcal{T}$, there exists a maximal, nonempty set $L \subseteq \{k \mid s \overset{k}{\dashrightarrow},\ s \in \mathcal{T}\}$ of pairwise unifiable labels such that $k \in L$. So there exists a transition $\mathcal{T} \overset{\ell}{\dashrightarrow} \mathcal{T}_\ell$ where $\ell = lub(L)$, which in particular means that $k \sqsubseteq \ell$. Moreover, by the definition of $\mathcal{T}_\ell$, we have that $s' \in \mathcal{T}_\ell$, hence $(s', \mathcal{T}_\ell) \in R$.

   Second, assume that $\mathcal{T} \overset{\ell}{\longrightarrow} \mathcal{T}_\ell$. Then we know that for all $t \in \mathcal{T}$ there is $t \overset{k}{\longrightarrow} t'$ such that $k \sqsubseteq \ell$ and $t' \in \mathcal{T}_\ell$. Since we know that $s \in \mathcal{T}$, it follows that there is $s \overset{k}{\longrightarrow} s'$ such that $k \sqsubseteq \ell$ and $s' \in \mathcal{T}_\ell$, hence $(s', \mathcal{T}_\ell) \in R$.

3. Let $D \in \mathcal{M}_{(K,\sqsubseteq)}$ be a deterministic LSMTS and assume that $S \leq_m D$ witnessed by the relation R. We want to show that $\mathcal{D}(S) \leq_m D$. We define a relation $R' \subseteq \mathcal{D}(S) \times D$ by

$$(\mathcal{T}, d) \in R' \qquad \text{if and only if} \qquad \emptyset \neq \mathcal{T} \subseteq \{s \in S \mid (s, d) \in R\}.$$

We show that R' is a relation witnessing $\mathcal{D}(S) \leq_m D$. Clearly, $(\{s_0\}, d_0) \in R'$ for the corresponding initial states. Let $(\mathcal{T}, d) \in R'$.

- First, assume that $\mathcal{T} \overset{\ell}{\dashrightarrow} \mathcal{T}_\ell$ and $\ell = lub(L)$ for some maximal, nonempty set $L \subseteq \{k \mid s \overset{k}{\dashrightarrow},\ s \in \mathcal{T}\}$. We want to show that there exists $d \overset{\ell'}{\dashrightarrow} d'$ such that $\ell \sqsubseteq \ell'$ and $(\mathcal{T}_\ell, d') \in R'$. Let

$$\mathcal{S} = \{s \in \mathcal{T} \mid s \overset{k}{\dashrightarrow},\ k \in L\}$$

which is nonempty since L is nonempty and it is a subset of $\{k \mid s \overset{k}{\dashrightarrow},\ s \in \mathcal{T}\}$. From the assumption $(\mathcal{T}, d) \in R'$, we know that for all $s \in \mathcal{S}$ it holds that $(s, d) \in R$ by the definition of R'. Let

$$\Delta = \{s \overset{k}{\dashrightarrow} s' \mid s \in \mathcal{S} \text{ and } k \in L\}.$$

For every transition $s \overset{k}{\dashrightarrow} s'$ in Δ, it follows from $(s, d) \in R$ that there exists $d \overset{k'}{\dashrightarrow} d_{k'}$ in D such that $k \sqsubseteq k'$ and $(s', d_{k'}) \in R$. Let L' denote the set of all such k'. Since L is a set of pairwise unifiable labels and for every $k' \in L'$ there is some $k \in L$ such that $k \sqsubseteq k'$, we know that L' is a set of pairwise unifiable labels, too. For every $k' \in L'$ we have a transition $d \overset{k'}{\dashrightarrow} d_{k'}$ for some $d_{k'} \in D$. From the determinism of D, it follows that there exists $d \overset{\ell'}{\dashrightarrow} d'$ such that $d' = d_{k'}$ and $\ell' = k'$ for all $k' \in L'$. Hence $k \sqsubseteq \ell'$ for all $k \in L$. Since ℓ is the least upper bound of L, we can conclude that $\ell \sqsubseteq \ell'$. Moreover, it holds that $(\mathcal{T}_\ell, d') \in R'$ because $\mathcal{T}_\ell = \{s' \in S \mid s \overset{k}{\dashrightarrow} s',\ s \in \mathcal{T},\ k \in L\}$ is nonempty, and for every $s' \in \mathcal{T}_\ell$ it holds that $(s', d') \in R$.

- Second, assume that $d \xrightarrow{\ell'} d'$. By the definition of R' we know that $(s, d) \in R$ for all $s \in \mathcal{T}$. Then it follows that, for all $s \in \mathcal{T}$, we have $s \xrightarrow{k} s'$ such that $k \sqsubseteq \ell'$ and $(s', d') \in R$. Note that $\mathcal{T}$ is nonempty, thus there is some maximal, nonempty set $L \subseteq \{k \mid s \overset{k}{\dashrightarrow},\ s \in \mathcal{T}\}$ of pairwise unifiable labels such that $k \in L$ for all labels k where $s \overset{k}{\dashrightarrow}$ and $k \sqsubseteq \ell'$. This implies that there exists $\mathcal{T} \xrightarrow{\ell} \mathcal{T}_\ell$ with $\ell = lub(L)$. Now, we have to show that $\ell \sqsubseteq \ell'$. Let $k \in L$, then there exists $s \overset{k}{\dashrightarrow} s'$ with $s \in \mathcal{T}$, and from $(s, d) \in R$ it follows that $k \sqsubseteq \ell'$ by the determinism of D, and $(s', d') \in R$. Since ℓ is the least upper bound of L, it follows that $\ell \sqsubseteq \ell'$. Finally, we show that $(\mathcal{T}_\ell, d') \in R'$. The set $\mathcal{T}_\ell$ is clearly nonempty, and moreover, it holds that for every $s' \in \mathcal{T}_\ell$ we have $(s', d') \in R$. ■

We can use the above theorem to prove that given two LSMTSs S and T, whenever S thoroughly refines T, then $\mathcal{D}(S)$ modally (syntactically) refines $\mathcal{D}(T)$.

Corollary 30. *Let $(K, \sqsubseteq)$ be a well-formed label-set for which the label refinement $\sqsubseteq$ is complete. Let $S, T \in \mathcal{M}_{(K,\sqsubseteq)}$ be locally consistent LSMTSs. If $S \leq_t T$ then $\mathcal{D}(S) \leq_m \mathcal{D}(T)$.*

Proof. By Theorem 29 we know that $T \leq_m \mathcal{D}(T)$ and by Theorem 17 also $T \leq_t \mathcal{D}(T)$. By transitivity and the assumption $S \leq_t T$, it follows that $S \leq_t \mathcal{D}(T)$, which implies $S \leq_m \mathcal{D}(T)$ by completeness of refinement (Theorem 26). By minimality of $\mathcal{D}(S)$ we can conclude that $\mathcal{D}(S) \leq_m \mathcal{D}(T)$. ■

3. Specification Theory

In order to apply label-structured modal transition systems as a specification formalism for software components, we need to define several operators on LSMTSs essential for any specification theory supporting compositional reasoning. First, *structural composition* (or *parallel composition*) allows us to combine interacting specifications. Second, *logical composition* (or *conjunction*) of two or more specifications expresses the greatest specification satisfying all the requirements of the given set of specifications. And third, a *quotient operator* which is dual to parallel composition: given a specification T expressing a requirement that needs to be implemented, and a specification S of the components that already exist, the quotient of T by S is the smallest specification which, together with S, satisfies T. All these operators are an important part of any compositional specification theory.

3.1. Operators on Labels and their Product

Operators on LSMTSs naturally involve different ways of combining labels. In this subsection, we introduce *label operators* on well-formed label-sets which will become crucial ingredients for the operators on LSMTSs introduced later on. In the text to follow, label operators are consistently denoted by symbols in circles.

Definition 31 (Label operator)**.** A (binary) *label operator* on a label-set $(K, \sqsubseteq)$ is a partial function $\odot : K \times K \rightharpoonup K$.

A label operator $\odot$ on $(K, \sqsubseteq)$ is called *commutative* if $k \odot \ell$ is defined iff $\ell \odot k$ is defined, and if they are defined then $k \odot \ell = \ell \odot k$. The operator is *associative* if $k \odot (\ell \odot m)$ is defined iff $(k \odot \ell) \odot m$, and if they are defined then $k \odot (\ell \odot m) = (k \odot \ell) \odot m$.

We can also form products of label operators which are operators on the product of their label-sets.

Definition 32 (Product of label operators)**.** Given two well-formed label-sets $(K_1, \sqsubseteq_1)$ and $(K_2, \sqsubseteq_2)$, and two label operators $\odot_1$ and $\odot_2$, the product of $\odot_1$ and $\odot_2$ is given by

the label operator $\odot_1 \times \odot_2$ on $K_1 \otimes K_2$ which is defined as follows:

$$(k_1, k_2)(\odot_1 \times \odot_2)(\ell_1, \ell_2) = \begin{cases} (k_1 \odot_1 \ell_1, k_2 \odot_2 \ell_2) & \text{if } \bot \neq k_i \odot_i \ell_i \text{ is defined for } i \in \{1,2\} \\ \bot & \text{if } k_i \odot_i \ell_i \text{ is defined for } i \in \{1,2\} \\ & \text{and either } k_1 \odot \ell_1 = \bot \text{ or } k_2 \odot \ell_2 = \bot \\ \text{undefined} & \text{otherwise} \end{cases}$$

It is easy to see that if two label operators, $\odot_1$ and $\odot_2$, are commutative and associative then so is the product operator $\odot_1 \times \odot_2$.

3.2. Parallel Composition

We start with the parallel composition, an operator that reflects the standard structural composition of implementations at the specification level.

Two LSMTSs with the same well-formed label-set $(K, \sqsubseteq)$ can be structurally composed with respect to a label operator $\oplus$ on $(K, \sqsubseteq)$. Two transitions (in different parallel components) labelled with $k \in K$ and $\ell \in K$ can synchronize if $k \oplus \ell$ is defined. The synchronized transition is then labelled with the label $k \oplus \ell$.

The desired property of parallel composition, crucial for any compositional specification theory, is called compositional refinement. It allows for a step-wise refinement of individual specifications while their parallel composition is guaranteed to refine the parallel composition of the original specifications. However, in order to achieve this, we have to impose a natural requirement on the label operator $\oplus$ for composing labels.

Definition 33 (Compositional label operator)**.** A label operator $\oplus$ on a well-formed label-set $(K, \sqsubseteq)$ is *compositional* if whenever $k' \sqsubseteq k$ and $\ell' \sqsubseteq \ell$ then $k' \oplus \ell'$ is defined if and only if $k \oplus \ell$ is defined, and in the positive case moreover $k' \oplus \ell' \sqsubseteq k \oplus \ell$.

As expected, compositionality of label operators is preserved under their products.

Proposition 34. *If $\oplus_1$ and $\oplus_2$ are compositional label operators on well-formed label-sets $(K_1, \sqsubseteq_1)$ and $(K_2, \sqsubseteq_2)$ respectively, then $\oplus_1 \times \oplus_2$ is a compositional label operator on $(K_1, \sqsubseteq_1) \otimes (K_2, \sqsubseteq_2)$.*

The actual definition of $\oplus$ depends on the interpretation of parallel composition for the modelled quantity. We present several possible definitions of $\oplus$ for some of the examples seen so far. All of the defined operators are compositional label operators and they can be further combined using the product construction.

Example 35. In Example 5 we have instantiated LSMTSs to modal transition systems labelled with actions, with well-formed label-set $K_{action} = (\Sigma \cup \{\bot\}, \sqsubseteq)$ for a finite set of actions Σ. The label operator for K_{action} can be defined as follows, depending on the desired synchronization scheme.

- *Synchronization by shared actions:*

$$a \oplus b = \begin{cases} a & \text{if } a = b \neq \bot \\ \bot & \text{if } a = \bot \text{ or } b = \bot \\ \text{undefined} & \text{otherwise} \end{cases}$$

- *Complete interleaving:*

$$a \oplus b = \begin{cases} a & \text{if } a \neq \bot \text{ and } b = e \\ b & \text{if } a = e \text{ and } b \neq \bot \\ \bot & \text{if } a = \bot \text{ or } b = \bot \\ \text{undefined} & \text{otherwise} \end{cases}$$

  Here we assume the existence of a special action $e \in K$ s.t. $s \overset{e}{\dashrightarrow} s$ for every state s.

□

Example 36. For weighted modal automata (see Example 7), the definition of $\oplus$ depends on how we want to interpret the weights. If the weights on transitions model e.g. *costs* (or *energy consumption*) then the composition operator may be defined as the sum of intervals:

$$k \oplus \ell = \begin{cases} [i_1 + i_2, j_1 + j_2] & \text{if } k = [i_1, j_1] \text{ and } \ell = [i_2, j_2] \\ \bot & \text{if } k = \bot \text{ or } \ell = \bot \end{cases}$$

Note that this label operator is total, so every transition in a weighted modal automaton will synchronize with each transition in the other automaton. Other options may include taking the interval intersection as the composition operator, should the weights represent e.g. (discrete) time intervals in which a transition can be executed. □

For the rest of this subsection, let us fix a well-formed label-set $(K, \sqsubseteq)$ with a compositional label operator $\oplus$ on it.

Definition 37 (Parallel composition)**.** Let $S, T \in \mathcal{M}_{(K,\sqsubseteq)}$ be two LSMTSs such that $S = (S, s_0, (K, \sqsubseteq), \dashrightarrow_S, \longrightarrow_S)$ and $T = (T, t_0, (K, \sqsubseteq), \dashrightarrow_T, \longrightarrow_T)$. The parallel composition of S and T is defined as the LSMTS

$$S \| T = (S \times T, (s_0, t_0), (K, \sqsubseteq), \dashrightarrow, \longrightarrow)$$

where the transition relations $\dashrightarrow$ and $\longrightarrow$ are defined by the following rules:

$$\frac{s \overset{k}{\dashrightarrow}_S s' \quad t \overset{\ell}{\dashrightarrow}_T t' \quad k \oplus \ell \text{ is defined}}{(s,t) \overset{k \oplus \ell}{\dashrightarrow} (s',t')} \qquad \frac{s \overset{k}{\longrightarrow}_S s' \quad t \overset{\ell}{\longrightarrow}_T t' \quad k \oplus \ell \text{ is defined}}{(s,t) \overset{k \oplus \ell}{\longrightarrow} (s',t')}$$

As we assumed that the label operator $\oplus$ is compositional, we get the property of compositional refinement, also called independent implementability [dAH05]. In other words, modal refinement is a precongruence with respect to parallel composition. This is formalized in the following theorem.

Theorem 38 (Independent Implementability)**.** *Let $S, S', T, T' \in \mathcal{M}_{(K,\sqsubseteq)}$ be LSMTSs and let $\oplus$ be a compositional label operator on $(K, \sqsubseteq)$. If $S' \leq_m S$ and $T' \leq_m T$ then $S'\|T' \leq_m S\|T$.*

Proof. Assume that R_1 is a relation showing $S' \leq_m S$ and R_2 is a relation showing $T' \leq_m T$. We define a relation $R \subseteq (S' \times T') \times (S \times T)$ by $((s', t'), (s, t)) \in R$ if and only if $(s', s) \in R_1$ and $(t', t) \in R_2$. We show that R witnesses $S'\|T' \leq_m S\|T$.

Obviously $((s'_0, t'_0), (s_0, t_0)) \in R$ where s_0, s'_0, t_0, t'_0 are the initial states of S, S', T, T', respectively. Let $((s', t'), (s, t)) \in R$.

1. Assume $(s', t') \overset{k' \oplus \ell'}{\dashrightarrow} (\hat{s}', \hat{t}')$. By the rule of parallel composition, we have $s' \overset{k'}{\dashrightarrow} \hat{s}'$ and $t' \overset{\ell'}{\dashrightarrow} \hat{t}'$. Then, from $(s', s) \in R_1$ and $(t', t) \in R_2$ it follows that there exist $s \overset{k}{\dashrightarrow} \hat{s}$ and $t \overset{\ell}{\dashrightarrow} \hat{t}$ such that $k' \sqsubseteq k$, $\ell' \sqsubseteq \ell$, $(\hat{s}', \hat{s}) \in R_1$, and $(\hat{t}', \hat{t}) \in R_2$. From the fact that $\oplus$ is a compositional label operator it follows that $k' \oplus l' \sqsubseteq k \oplus \ell$, and then $(s, t) \overset{k \oplus \ell}{\dashrightarrow} (\hat{s}, \hat{t})$ and $((\hat{s}', \hat{t}'), (\hat{s}, \hat{t})) \in R$.

2. Assume $(s, t) \overset{k \oplus \ell}{\longrightarrow} (\hat{s}, \hat{t})$. By the rule of parallel composition, we have $s \overset{k}{\longrightarrow} \hat{s}$ and $t \overset{\ell}{\longrightarrow} \hat{t}$. Then, from $(s', s) \in R_1$ and $(t', t) \in R_2$ it follows that there exist $s' \overset{k'}{\longrightarrow} \hat{s}'$ and $t \overset{\ell'}{\longrightarrow} \hat{t}'$ such that $k' \sqsubseteq k$, $\ell' \sqsubseteq \ell$, $(\hat{s}', \hat{s}) \in R_1$, and $(\hat{t}', \hat{t}) \in R_2$. From the compositionality of the label operator it follows that $k' \oplus \ell' \sqsubseteq k \oplus \ell$, and then $(s', t') \overset{k' \oplus \ell'}{\longrightarrow} (\hat{s}', \hat{t}')$ and $((\hat{s}', \hat{t}'), (\hat{s}, \hat{t})) \in R$.

■

Clearly, if $\oplus$ is commutative and associative, then so is the parallel composition (up to isomorphism).

3.3. Conjunction

Different component requirements can be often specified by independent teams. The issue of dealing with the aspects of multiple viewpoints/properties is thus essential. It should be possible to represent several specifications (viewpoints) for the same implementation and to combine them in a logical manner. This is the objective of the *conjunction* operation.

Two LSMTSs with the same label-set $(K, \sqsubseteq)$ can be conjoined with respect to a label operator $\oslash$ on $(K, \sqsubseteq)$. We first state a necessary condition on $\oslash$ such that the conjunction operator yields the greatest lower bound with respect to the modal refinement relation on LSMTSs.

Definition 39 (Greatest lower bound operator)**.** A commutative label operator $\oslash$ on a well-formed set $(K, \sqsubseteq)$ is a *greatest lower bound operator* if the following is satisfied:

1. if $k \oslash \ell$ is defined, then $k \oslash \ell \sqsubseteq k$ and $k \oslash \ell \sqsubseteq \ell$,
2. if $m \neq \bot$, $m \sqsubseteq k$ and $m \sqsubseteq \ell$, then $k \oslash \ell$ is defined and $m \sqsubseteq k \oslash \ell$.

As in the case of the compositional label operator, it is easy to see that greatest lower bound operators are preserved by the product construction.

Proposition 40. *Let $\oslash_1$ and $\oslash_2$ be greatest lower bound operators on $(K_1, \sqsubseteq_1)$ and $(K_2, \sqsubseteq_2)$, respectively. Then $\oslash_1 \times \oslash_2$ is a greatest lower bound operator on the product $(K_1, \sqsubseteq_1) \otimes (K_2, \sqsubseteq_2)$.*

Again, the actual definition of $\oslash$ depends on the interpretation of conjunction for the modelled quantity.

Example 41. Conjoining labels in $K_{action} = (\Sigma \cup \{\bot\}, \sqsubseteq)$, for a finite set of actions Σ (see Example 5), can be defined as follows:

$$a \oslash b = \begin{cases} a & \text{if } a = b,\ a \neq \bot,\ b \neq \bot \\ \bot & \text{if } a = \bot \text{ or } b = \bot \\ \text{undefined} & \text{otherwise} \end{cases}$$

□

Example 42. For the case of weighted modal automata (see Example 7), conjunction $\oslash$ can be defined as the intersection of the intervals, assuming that we consider the cost (energy) interpretation.

$$k \oslash \ell = \begin{cases} k \cap \ell & \text{if } k \neq \bot,\ \ell \neq \bot \text{ and } k \cap \ell \neq \emptyset \\ \bot & \text{otherwise} \end{cases}$$

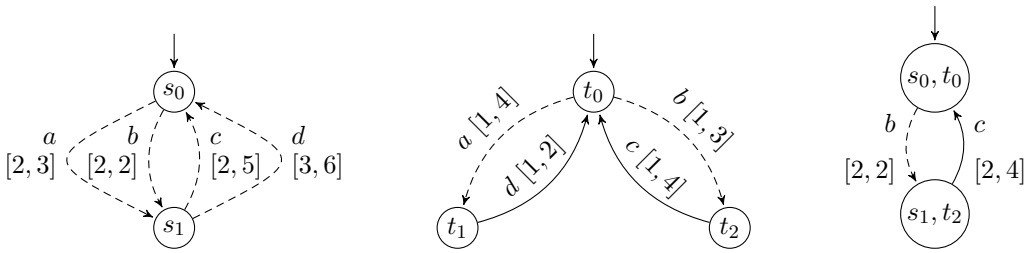
□

It is easy to see that the label operators defined in Examples 41 and 42 are greatest lower bound operators on their respective label-sets.

Let us fix a well-formed label-set $(K, \sqsubseteq)$ and some greatest lower bound operator $\oslash$ on $(K, \sqsubseteq)$ for the rest of this subsection.

Definition 43 (Conjunction)**.** Let $S, T \in \mathcal{M}_{(K,\sqsubseteq)}$ be two LSMTSs such that $S = (S, s_0, (K, \sqsubseteq), \dashrightarrow_S, \longrightarrow_S)$ and $T = (T, t_0, (K, \sqsubseteq), \dashrightarrow_T, \longrightarrow_T)$. The *conjunction of S and T* is defined by the LSMTS $S \wedge T = (S \times T, (s_0, t_0), (K, \sqsubseteq), \dashrightarrow, \longrightarrow)$ where the transition relations $\dashrightarrow$ and $\longrightarrow$ are defined by the following rules.

$$\frac{s \xrightarrow{k}_S s' \quad t \overset{\ell}{\dashrightarrow}_T t' \quad k \oslash \ell \text{ is defined}}{(s,t) \xrightarrow{k \oslash \ell} (s', t')} \qquad \frac{s \overset{k}{\dashrightarrow}_S s' \quad t \xrightarrow{\ell}_T t' \quad k \oslash \ell \text{ is defined}}{(s,t) \xrightarrow{k \oslash \ell} (s', t')}$$



(a) S (b) T (c) $\rho(S \wedge T)$

Figure 8.: Pruned conjunction of two LSMTSs S and T

$$\frac{s \overset{k}{\dashrightarrow}_S s' \quad t \overset{\ell}{\dashrightarrow}_T t' \quad k \otimes \ell \text{ is defined}}{(s,t) \overset{k \otimes \ell}{\dashrightarrow} (s',t')}$$

$$\frac{s \xrightarrow{k}_S s' \quad \big(k \otimes \ell \text{ is not defined for any } \ell \text{ such that } t \overset{\ell}{\dashrightarrow}_T\big)}{(s,t) \xrightarrow{\perp} (s,t)}$$

$$\frac{t \xrightarrow{\ell}_T t' \quad \big(k \otimes \ell \text{ is not defined for any } k \text{ such that } s \overset{k}{\dashrightarrow}_S\big)}{(s,t) \xrightarrow{\perp} (s,t)}$$

Clearly, conjunction $\wedge$ on LSMTSs is commutative (up to isomorphism) as $\otimes$ is commutative, and moreover, if $\otimes$ is an associative label operator, then so is conjunction.

Example 44. An example for conjoining specifications is given in Figure 8. Here S and T are LSMTSs with the label-set $K_{action} \otimes K_{weighted}$. Note that the state (s_1, t_1) does not appear in $\rho(S \wedge T)$ since it is locally inconsistent in $S \wedge T$; the LSMTS T requires a transition labelled with the action d and the weight interval $[1,2]$, however, S only allows d with the weight interval $[3,6]$. □

We now propose a notion of determinism that is dedicated to the conjunction operation. This definition shall be used later on to prove that conjunction is the greatest lower bound with respect to the modal refinement ordering.

Definition 45 ($\otimes$-determinism)**.** A LSMTS $S \in \mathcal{M}_{(K,\sqsubseteq)}$ is $\otimes$-*deterministic* if for all may transitions $s \overset{k'}{\dashrightarrow} s'$ and $s \overset{k''}{\dashrightarrow} s''$ in S with labels $k', k'' \in K \setminus \{\perp\}$, whenever there is an $\ell \in K \setminus \{\perp\}$ such that both $k' \otimes \ell$ and $k'' \otimes \ell$ are defined, then $k' = k''$ and $s' = s''$.

It is easy to see that if $\otimes$ is associative, then the construction for conjunction presented in Definition 43 preserves $\otimes$-determinism. The reader may also observe that for modal transition systems with the label-set K_{action}, the notions of $\otimes$-determinism and determinism (as defined in Section 2) coincide. In this case, we note that the determinization algorithm proposed in Section 2.5 can be applied to compute minimal deterministic LSMTSs of non-deterministic ones.

Under the assumption of $\otimes$-determinism, the conjunction construction yields the greatest lower bound with respect to modal refinement, as stated in the following theorem.

Theorem 46 (Greatest lower bound of conjunction)**.** *Let $S, T, U \in \mathcal{M}_{(K,\sqsubseteq)}$ be locally consistent LSMTSs such that S and T are $\oslash$-deterministic and $S \wedge T$ is consistent. Assume that $\oslash$ is a greatest lower bound label operator. Then*

1. *$\rho(S \wedge T) \leq_m S$ and $\rho(S \wedge T) \leq_m T$, and*
2. *if $U \leq_m S$ and $U \leq_m T$, then $U \leq_m \rho(S \wedge T)$.*

Proof. 1. It suffices to show that $\rho(S \wedge T) \leq_m S$, the other assertion is symmetric. We define a relation $R \subseteq (S \times T) \times S$ by $R = \{((s,t), s) \mid s \in S, t \in T\}$. We will show that R is a relation witnessing $\rho(S \wedge T) \leq_m S$. Clearly $((s_0, t_0), s_0) \in R$ where s_0 is the initial state of S and t_0 is the initial state of T. Let $((s,t), s) \in R$.

Let $(s,t) \overset{k \oslash \ell}{\dashrightarrow} (s', t')$. Since $\rho(S \wedge T)$ does not contain any transitions labelled with $\bot$, we know that $k \oslash \ell \neq \bot$. Then there are transitions $s \overset{k}{\dashrightarrow} s'$ and $t \overset{\ell}{\dashrightarrow} t'$. By assumption we know $k \oslash \ell \sqsubseteq k$, and from the definition of R we can conclude $((s', t'), s') \in R$.

Now, let $s \xrightarrow{k} s'$. By local consistency of S we can assume that $k \neq \bot$. Suppose that T does not have any $t \overset{\ell}{\dashrightarrow} t'$ such that $k \oslash \ell$ is defined, then $(s,t) \xrightarrow{\bot} (s', t)$ contradicting the local consistency of $\rho(S \wedge T)$. So there exists $t \overset{\ell}{\dashrightarrow} t'$ such that $k \oslash \ell$ is defined and then $(s,t) \xrightarrow{k \oslash \ell} (s', t')$. By the assumption about the label operator we know that $k \oslash \ell \sqsubseteq k$. By the definition of R we get $((s', t'), s') \in R$.

2. We can assume a relation R_1 witnessing $U \leq_m S$ and a relation R_2 witnessing $U \leq_m T$. We define a relation $R \subseteq U \times (S \times T)$ by

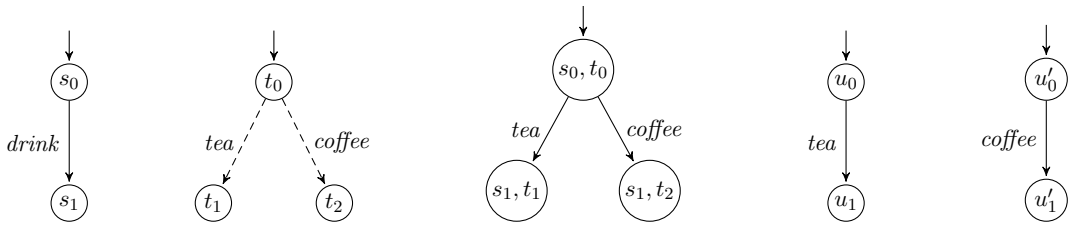
$$R = \{(u, (s,t)) \mid (u,s) \in R_1 \text{ and } (u,t) \in R_2\}.$$

We show that R is witnessing $U \leq_m \rho(S \wedge T)$. Clearly $(u_0, (s_0, t_0)) \in R$ for the initial states. Let $(u, (s,t)) \in R$.

Assume that $u \overset{m}{\dashrightarrow} u'$. Then there exists $s \overset{k}{\dashrightarrow} s'$ such that $m \sqsubseteq k$, and $t \overset{\ell}{\dashrightarrow} t'$ such that $m \sqsubseteq \ell$. By Definition 39 part 2. we get that $k \oslash \ell$ is defined and $m \sqsubseteq k \oslash \ell$, hence $(s,t) \overset{k \oslash \ell}{\dashrightarrow} (s', t')$, and $(u', (s', t')) \in R$ by the definition of R.

Assume that $(s,t) \xrightarrow{k \oslash \ell} (s', t')$. By local consistency of $\rho(S \wedge T)$ we can assume that $k \oslash \ell \neq \bot$. Then (w.l.o.g.) there exist $s \xrightarrow{k} s'$ with $k \neq \bot$, and $t \overset{\ell}{\dashrightarrow} t'$ with $\ell \neq \bot$ thanks to local consistency of S and T. It follows from $(u,s) \in R_1$ that there exists $u \xrightarrow{m} u'$ such that $(u', s') \in R_1$ and $m \sqsubseteq k$. Local consistency of U implies $m \neq \bot$. We have to show that $(u', t') \in R_2$. From $(u,t) \in R_2$ it follows that there exists $t \overset{\ell'}{\dashrightarrow} t''$ such that $m \sqsubseteq \ell'$ and $(u', t'') \in R_2$. Now, by $m \neq \bot$, $m \sqsubseteq k$ and $m \sqsubseteq \ell'$ we know that $k \oslash \ell'$ is defined (Definition 39 part 2.). From $\oslash$-determinism of T we get $\ell = \ell'$ and $t' = t''$. It follows that $(u', t') \in R_2$ and thus $(u', (s', t')) \in R$.

■



(a) S (b) T (c) $S \wedge T = \rho(S \wedge T)$ (d) U (e) U'

Figure 9.: Conjunction of two LSMTSs specifying a vending machine

Corollary 47. *Let $S, T, U \in \mathcal{M}_{(K,\sqsubseteq)}$ be locally consistent LSMTSs such that S and T are $\oslash$-deterministic and $S \wedge T$ is consistent. Then $U \leq_m S$ and $U \leq_m T$ if and only if $U \leq_m \rho(S \wedge T)$. In particular, $[\![S \wedge T]\!] = [\![S]\!] \cap [\![T]\!]$.*

Proof. The implication from left to right is exactly the second part of Theorem 46. The other direction follows from the first part of Theorem 46 and the transitivity of $\sqsubseteq$. The additional assertion $[\![S \wedge T]\!] = [\![S]\!] \cap [\![T]\!]$ follows from $U \leq_m S$ and $U \leq_m T$ if and only if $U \leq_m \rho(S \wedge T)$ if we take U as an implementation, and by Proposition 22 showing that $[\![\rho(S \wedge T)]\!] = [\![S \wedge T]\!]$. ■

For maximality of the conjunction (see Theorem 46, 2.), $\oslash$-determinism is necessary. To see this, consider the example shown in Figure 9 that shows two specifications of a vending machine with the label-set $K_{machine}$ from Example 6. The LSMTS S requires *drink*, and T allows *tea* and *coffee*. For the conjunction $S \wedge T$ we take $\oslash$ as the obvious greatest lower bound for $K_{machine}$ for which $drink \oslash tea = tea$ and $drink \oslash coffee = coffee$. Note that T is *not* $\oslash$-deterministic in this case. It is easy to see that $(S \wedge T) \leq_m S$ and $(S \wedge T) \leq_m T$, however U and U' are both refining S and T, but $U \not\leq_m (S \wedge T) = \rho(S \wedge T)$. With a small extension of the proof for modal transition systems [DLLW10], it can easily be proved that there does not exist any LSMTS which is the greatest lower bound for S and T but the construction of conjunction in this case is at least safe.

3.4. Quotient

An essential operator in a complete specification theory is the one of *quotienting*. It allows for factoring out behaviours from a larger component. Given two component specifications S and T, the quotient of T by S, written $T \setminus\!\!\setminus S$, is a specification of exactly those components that when composed with S refine T. In other words, the quotient is the largest specification that can be composed with S and still refines T.

As expected, we have to first state the required property for label operators used for quotienting.

Definition 48 (Dual label operators)**.** Let $\ominus$ and $\oplus$ be two label operators on a given well-formed label-set $(K, \sqsubseteq)$. We say that the operator $\ominus$ is a *dual label operator to* $\oplus$ if $m \sqsubseteq \ell \ominus k$ if and only if $k \oplus m \sqsubseteq \ell$.

Example 49. Quotienting labels in $K_{action} = (\Sigma \cup \{\bot\}, \sqsubseteq)$ for a finite set of actions Σ as introduced in Example 5, together with the $\oplus$ operator for synchronization by shared actions given in Example 35 can be defined as identical to $\oplus$, namely:

$$a \ominus b = \begin{cases} a & \text{if } a = b \neq \bot \\ \bot & \text{if } a = \bot \text{ or } b = \bot \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Clearly, $\ominus$ is a dual label operator to $\oplus$. □

Example 50. A more interesting example of $\ominus$ in $K_{weighted}$ (see Example 7) dual to the composition operator $\oplus$ from Example 36 that sums up weight intervals is given by

$$[a,b] \ominus [a',b'] = \begin{cases} [a-a', b-b'] & \text{if } [a-a', b-b'] \in K_{weighted} \\ \bot & \text{otherwise.} \end{cases}$$

□

Let us for the rest of this subsection fix two label operators $\ominus$ and $\oplus$ such that $\ominus$ is a dual label operator to $\oplus$ on a well-formed label-set $(K, \sqsubseteq)$.

Definition 51 (Quotient)**.** Let $S, T \in \mathcal{M}_{(K,\sqsubseteq)}$ be two LSMTS such that $S = (S, s_0, (K, \sqsubseteq), \dashrightarrow_S, \longrightarrow_S)$ and $T = (T, t_0, (K, \sqsubseteq), \dashrightarrow_T, \longrightarrow_T)$. The *quotient of T by S* is defined as $T \setminus\!\!\setminus S = \big((T \times S) \cup \{\mathsf{u}\}, (t_0, s_0), (K, \sqsubseteq), \dashrightarrow, \longrightarrow\big)$ where u is a new state modelling a universal state, and the transition relations $\dashrightarrow$ and $\longrightarrow$ are defined by the following rules:

$$\frac{t \overset{\ell}{\dashrightarrow}_T t' \quad s \overset{k}{\dashrightarrow}_S s' \quad \ell \ominus k \text{ is defined}}{(t,s) \overset{\ell \ominus k}{\dashrightarrow} (t',s')} \qquad \frac{t \overset{\ell}{\longrightarrow}_T t' \quad s \overset{k}{\longrightarrow}_S s' \quad \ell \ominus k \text{ is defined}}{(t,s) \overset{\ell \ominus k}{\longrightarrow} (t',s')}$$

$$\frac{t \overset{\ell}{\longrightarrow}_T \quad \big(\ell \ominus k \text{ is not defined for any } k \text{ s.t. } s \overset{k}{\longrightarrow}_S\big)}{(t,s) \overset{\bot}{\longrightarrow} (t,s)}$$

$$\frac{m \in K \text{ and } m \oplus k \text{ is not defined for any } s \overset{k}{\dashrightarrow}_S}{(t,s) \overset{m}{\dashrightarrow} \mathsf{u}}$$

$$\frac{m \in K}{\mathsf{u} \overset{m}{\dashrightarrow} \mathsf{u}}$$

The first two rules presented above are derived from the two rules of Definition 37, while the third one captures the inconsistency present when the larger system T has a must transition that cannot be mimicked by the smaller system S in parallel with any transition from the quotient. To achieve maximality of the quotient, we introduce a universal state u that allows for an arbitrary behaviour. Any allowed behaviour not structurally composable with the allowed behaviour of the smaller system can be safely added to the quotient (leading to the state u) as this will not affect the parallel composition with the smaller system. This is captured by the fourth rule.

Example 52. An example of quotienting is shown in Figure 10. Both specifications T and S have as the label-set $K_{action} \otimes K_{weighted}$ where $K_{action} = (\{a, b, \bot\}, \sqsubseteq)$, and $\rho(T \setminus\!\setminus S)$ is the result of the pruned quotient of T by S with respect to the product of the corresponding label operators from Examples 49 and 50 where the resulting operators are summarized in Figure 11 (these operators are already defined by the product construction, we just present their combined definitions for clarity reasons). Thereby, a may transition under $[-\infty, \infty]$ between two states stands for all may transitions between those states under any label in $K_{weighted}$.

- The quotient $\rho(T \setminus\!\setminus S)$, in state (t_0, s_0), may do the action b with any weight (abbreviated by $[-\infty, \infty]$) and afterwards show arbitrary behaviour (reflected by the universal state) since S has no corresponding transition for action b in state s_0.
- Note that the state (t_3, s_3) is not present in the pruned quotient. If $\rho(T \setminus\!\setminus S)$ would allow for a transition labelled with a (which would be possible with the weight interval $[0, 0]$), then $(b, [1, 4]) \ominus (b, [0, 4])$ yields $\bot$ (on a must transition), turning (t_3, s_3) into an inconsistent state with no implementation.

□

The quotient $T \setminus\!\setminus S$ intends to synthesize the largest component that can be composed with S in order to refine T. In existing theories such as modal transition systems, this maximality property only holds when the specifications are deterministic. As for conjunction, we now propose a general notion of determinism for quotienting.

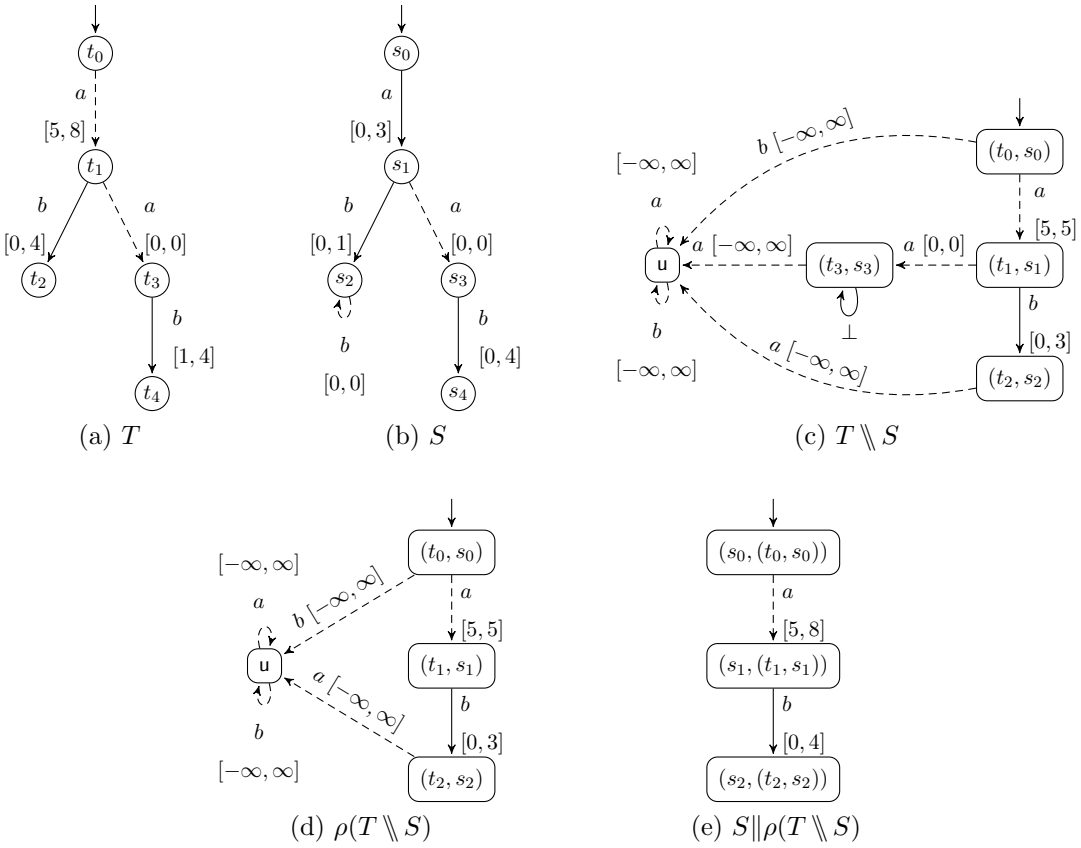
Definition 53 ($\oplus/\ominus$-determinism)**.** Let $S \in \mathcal{M}_{(K,\sqsubseteq)}$ be an LSMTS. We say that S is $\oplus/\ominus$-*deterministic* if, for any $k', k'' \in K \setminus \{\bot\}$,

1. whenever $s \overset{k'}{\dashrightarrow} s'$ and $s \overset{k''}{\dashrightarrow} s''$ and there exists $m \in K \setminus \{\bot\}$ such that $k' \oplus m$ and $k'' \oplus m$ are defined, then $k' = k''$ and $s' = s''$, and
2. whenever $s \xrightarrow{k'} s'$ and $s \xrightarrow{k''} s''$ and there exists $\ell \in K \setminus \{\bot\}$ such that $\ell \ominus k'$ and $\ell \ominus k''$ are defined, then $k' = k''$ and $s' = s''$.

The reader may again observe that for modal transition systems with the label-set K_{action}, the notions of $\oplus/\ominus$-determinism and determinism (as defined in Section 2) as well as $\oslash$-determinism coincide. Here, in case of non-deterministic LSMTSs, the determinization algorithm proposed in Section 2.5 can be applied to compute minimal deterministic versions of them.

Under the assumption of $\oplus/\ominus$-determinism, the quotient construction $T \setminus\!\setminus S$ yields the most general LSMTS that, composed with S, still refines T.

Theorem 54 (Soundness and maximality of quotient)**.** *Let $S, T, X \in \mathcal{M}_{(K,\sqsubseteq)}$ be locally consistent LSMTSs such that $T \setminus\!\setminus S$ is consistent. Assume that S is $\oplus/\ominus$-deterministic, and assume that $\ominus$ is the dual label operator to $\oplus$. Then $X \leq_m \rho(T \setminus\!\setminus S)$ if and only if $S \| X \leq_m T$.*



(a) T (b) S (c) $T \setminus\setminus S$

(d) $\rho(T \setminus\setminus S)$ (e) $S \| \rho(T \setminus\setminus S)$

Figure 10.: Quotienting of two LSMTSs

$$(a,i) \oplus (a',i') = \begin{cases} a, [l+l', r+r'] & \text{if } a = a' \neq \bot,\ i = [l,r],\ i' = [l',r'] \\ \bot & \text{if } a = \bot \text{ or } a' = \bot \text{ or } i = \bot \text{ or } i' = \bot \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$(a,i) \ominus (a',i') = \begin{cases} a, [l-l', r-r'] & \text{if } a = a' \neq \bot,\ i = [l,r],\ i' = [l',r'], \\ & [l-l', r-r'] \in K_{weighted} \\ \bot & \text{if } a = \bot \text{ or } a' = \bot \text{ or } i = \bot \text{ or } i' = \bot \text{ or} \\ & i = [l,r],\ i' = [l',r'] \text{ and } [l-l', r-r'] \notin K_{weighted} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Figure 11.: Label operators $\oplus$, $\ominus$ for quotienting with the label-set $K_{action} \otimes K_{weighted}$

Proof. "$\Longrightarrow$": We assume a relation R_1 witnessing $X \leq_m \rho(T \setminus\!\setminus S)$. We define a relation $R_2 \subseteq (S \times X) \times T$ by

$$R_2 = \{((s,x),t) \mid (x,(t,s)) \in R_1\}.$$

We show that R_2 is a relation witnessing $S \| X \leq_m T$. Clearly $((s_0,x_0),t_0) \in R_2$ for the corresponding initial states. Let $((s,x),t) \in R_2$.

1. Assume $(s,x) \overset{k \oplus m}{\dashrightarrow} (s',x')$. Then there exist $s \overset{k}{\dashrightarrow} s'$ and $x \overset{m}{\dashrightarrow} x'$. Since X is locally consistent, we know $m \neq \bot$. From $(x,(t,s)) \in R_1$ and $x \overset{m}{\dashrightarrow} x'$ it follows that there exists $(t,s) \overset{\ell \ominus k'}{\dashrightarrow} (t',s'')$ such that $m \sqsubseteq \ell \ominus k'$ and $(x',(t',s'')) \in R_1$. From $m \sqsubseteq \ell \ominus k'$ we get $k' \oplus m \sqsubseteq \ell$ by assumption. Moreover, we have $t \overset{\ell}{\dashrightarrow} t'$ and $s \overset{k'}{\dashrightarrow} s''$. Then, from $\oplus/\ominus$-determinism of S it follows $k = k'$ and $s' = s''$, and so $k \oplus m \sqsubseteq \ell$. Thus $(x',(t',s')) \in R_1$ and hence $((s',x'),t') \in R_2$.

2. Assume $t \xrightarrow{\ell} t'$. By local consistency of T we can assume that $\ell \neq \bot$. Suppose that there does not exist a transition $s \xrightarrow{k} s'$ such that $\ell \ominus k$ is defined, then $(t,s) \xrightarrow{\bot}$ which contradicts $(t,s) \in \rho(T \setminus\!\setminus S)$. It follows that there exists $s \xrightarrow{k} s'$ such that $\ell \ominus k$ is defined, and $(t,s) \xrightarrow{\ell \ominus k} (t',s')$. By a similar argument as above we know that $\ell \ominus k \neq \bot$. Then from $(x,(t,s)) \in R_1$ it follows that there exists $x \xrightarrow{m} x'$ such that $(x',(t',s')) \in R_1$ and $m \sqsubseteq \ell \ominus k$ which implies $k \oplus m \sqsubseteq \ell$. We can conclude that $(s,x) \xrightarrow{k \oplus m} (s',x')$ and $((s',x'),t') \in R_2$.

"$\Longleftarrow$": We assume a relation R_2 witnessing $S \| X \leq_m T$. We define a relation $R_1 \subseteq X \times (T \times S)$ by

$$R_1 = \{(x,(t,s)) \mid ((s,x),t) \in R_2\} \cup \{(x,\mathsf{u})\}.$$

We show that R_1 is a relation witnessing $X \leq_m \rho(T \setminus\!\setminus S)$. Clearly $(x_0,(t_0,s_0)) \in R_1$ for the corresponding initial states. First, let $(x,\mathsf{u}) \in R_1$, then clearly for every transition $x \overset{m}{\dashrightarrow} x'$, it holds again that $(x',\mathsf{u}) \in R_1$ since the universal state u allows arbitrary behaviour. Second, let $(x,(t,s)) \in R_1$.

1. Assume $x \overset{m}{\dashrightarrow} x'$. If there is no transition $s \overset{k}{\dashrightarrow} s'$ such that $k \oplus m$ is defined, then $(t,s) \overset{m}{\dashrightarrow} \mathsf{u}$, and in this case, $(x,\mathsf{u}) \in R_1$, and clearly $m \sqsubseteq m$ by reflexivity of $\sqsubseteq$. If there is a transition $s \overset{k}{\dashrightarrow} s'$ such that $k \oplus m$ is defined, then $(s,x) \overset{k \oplus m}{\dashrightarrow} (s',x')$. From $((s,x),t) \in R_2$ it follows that there exists $t \overset{\ell}{\dashrightarrow} t'$ such that $((s',x'),t') \in R_2$ and $k \oplus m \sqsubseteq \ell$, implying $m \sqsubseteq \ell \ominus k$. Hence $(t,s) \overset{\ell \ominus k}{\dashrightarrow} (t',s')$ and $(x',(t',s')) \in R_1$.

2. Assume $(t,s) \xrightarrow{\ell \ominus k} (t',s')$. From local consistency of $\rho(T \setminus\!\setminus S)$ we know $\ell \ominus k \neq \bot$. Then there are $t \xrightarrow{\ell} t'$ and $s \xrightarrow{k} s'$. From $((s,x),t) \in R_2$ we can conclude that there exists $(s,x) \xrightarrow{k' \oplus m} (s'',x')$ such that $k' \oplus m \sqsubseteq \ell$ and $((s'',x'),t') \in R_2$. Then there exist $s \xrightarrow{k'} s''$ and $x \xrightarrow{m} x'$. The fact that $k' \oplus m \sqsubseteq \ell$ implies $m \sqsubseteq \ell \ominus k'$ by the duality of the operators. From $\oplus/\ominus$-determinism of S it follows that $k = k'$ and $s' = s''$. Thus $m \sqsubseteq \ell \ominus k$ and $((s',x'),t') \in R_2$, hence $(x',(t',s')) \in R_1$. ∎

4. Logical Characterization

It was shown in [Lar89] that Hennessy-Milner logic [HM85] can be used as a logical characterization for modal refinement of modal transition systems (the reader may also consult [BG00]). In this section we shall extend this result to LSMTSs and study other related topics. For the rest of this section, we fix a well-formed label-set $(K, \sqsubseteq)$.

Let us first introduce LSHML, an extension of Hennessy-Milner logic (HML) that is interpreted over LSMTSs, taking into account their label structures. The syntax of the logic is given by the abstract syntax:

$$\varphi ::= \mathsf{true} \mid \mathsf{false} \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \langle k \rangle \psi \mid [k]\psi$$

where $k \in K$ is a label. We define the $\vee$-free fragment of LSHML as a set of formulae in LSHML not containing the disjunction operator.

Let $S \in \mathcal{M}_{(K,\sqsubseteq)}$ be an LSMTS. The satisfaction relation between a state $s \in S$ and a formula φ is defined inductively as follows.

$s \models \mathsf{true}$

$s \not\models \mathsf{false}$

$s \models \varphi_1 \wedge \varphi_2$ iff $S \models \varphi_1$ and $S \models \varphi_2$

$s \models \varphi_1 \vee \varphi_2$ iff $S \models \varphi_1$ or $S \models \varphi_2$

$s \models \langle k \rangle \varphi$ iff $\exists (s \xrightarrow{\ell} s') : [\![\ell]\!] \subseteq [\![k]\!]$ and $s' \models \varphi$

$s \models [k]\varphi$ iff $\forall (s \overset{\ell}{\dashrightarrow} s')$ s.t. $[\![\ell]\!] \cap [\![k]\!] \neq \emptyset : s' \models \varphi$

We write $S \models \varphi$ iff $s_0 \models \varphi$ where s_0 is the initial state of S.

Example 55. Consider the specification of a vending machine given in Figure 3c. The specification satisfies the property that after inserting a 1€ coin, a drink is guaranteed, as we have $s_0 \models [\mathit{1EURO}]\langle \mathit{drink} \rangle \mathsf{true}$. On the other hand, we are not guaranteed to receive a cup of tea if a coin is inserted as $s_0 \not\models [\mathit{coin}]\langle \mathit{tea} \rangle \mathsf{true}$. □

We are now ready to prove the soundness and completeness theorems for our logic. The following theorem ensures soundness of LSHML, i.e., if a formula holds for a specification, then it holds for any of its refinements.

Theorem 56 (Soundness)**.** *Let $T \in \mathcal{M}_{(K,\sqsubseteq)}$, and φ a LSHML-formula. Then*

$$T \models \varphi \implies \forall S \leq_m T : S \models \varphi \ .$$

Proof. Assume that $S \leq_m T$ and $T \models \varphi$. We prove by induction on the structure of φ that $S \models \varphi$ too. Let s_0 and t_0 be the initial states of S and T, respectively. The induction basis, where $\varphi = \mathsf{true}$ and $\varphi = \mathsf{false}$, is trivial.

$\varphi = \varphi_1 \wedge \varphi_2$. By the definition of $\models$ and then from the induction hypothesis.

$\varphi = \varphi_1 \vee \varphi_2$. As in the case above.

$\varphi = \langle k \rangle \psi$. From $T \models \langle k \rangle \psi$ it follows that there exists $t_0 \xrightarrow{\ell} t$ such that $[\![\ell]\!] \subseteq [\![k]\!]$ and $t \models \psi$. Since $S \leq_m T$ there exists $s_0 \xrightarrow{\ell'} s$ such that $\ell' \sqsubseteq \ell$ and $(s, S) \leq_m (t, T)$. By the induction hypothesis we get $s \models \psi$. From transitivity of $\sqsubseteq$ we have $[\![\ell']\!] \subseteq [\![k]\!]$ and therefore $S \models \varphi$.

$\varphi = [k] \psi$. Let $s_0 \overset{\ell}{\dashrightarrow} s$ such that $[\![k]\!] \cap [\![\ell]\!] \neq \emptyset$. Since $S \leq_m T$ we know that there exists $t_0 \overset{\ell'}{\dashrightarrow} t$ in T with $\ell \sqsubseteq \ell'$ and $(s, S) \leq_m (t, T)$. Clearly, $[\![k]\!] \cap [\![\ell']\!] \neq \emptyset$ and because $T \models [k] \psi$ we know that $t \models \psi$. By the induction hypothesis we get $s \models \psi$ and hence $S \models \varphi$.

■

We shall now focus on the issue of completeness. We consider two possible definitions:

1. *Completeness with respect to implementations*: if all implementations of a specification S satisfy a formula of the logic, then so does the specification S.
2. *Completeness with respect to modal refinement*: if all formulae satisfied by some specification S are satisfied also by another specification T, then $S \leq_m T$.

The latter, completeness with respect to modal refinement, is also known as logical characterization in the literature [Lar89].

We first study the completeness with respect to implementations and observe that LSHML-logic is not complete in this case.

Theorem 57. *The logic LSHML is incomplete with respect to implementations.*

Proof. Let T be an LSMTS consisting of a single transition $t_0 \overset{\bullet}{\dashrightarrow} t$ over the label-set $K_{unlabelled}$ from Example 4. Consider the formula $\varphi = \langle \bullet \rangle \mathsf{true} \vee [\bullet] \mathsf{false}$. Since there is no must transition from t_0 and at the same time there is a may transition, we get $t_0 \not\models \varphi$. On the other hand, any implementation of T either contains no transition at all (and then it satisfies $[\bullet]\mathsf{false}$) or it contains at least one outgoing transition (and then it satisfies $\langle \bullet \rangle \mathsf{true}$). Hence any implementation of T satisfies φ and we get the incompleteness result with respect to implementations. ■

Inspecting the proof of the above theorem, one can notice that it is the disjunction that breaks the completeness property. In fact, we can show completeness if we consider the $\vee$-free fragment of LSHML.

Theorem 58 (Completeness with respect to implementations for $\vee$-free LSHML)**.** *Let $T \in \mathcal{M}_{(K,\sqsubseteq)}$ be a locally consistent specification, and let φ be a $\vee$-free LSHML-formula. Then*

$$(\forall I \in [\![T]\!] : I \models \varphi) \quad \Longrightarrow \quad T \models \varphi \ .$$

Proof. We prove the contraposition. We show that for any $\vee$-free LSHML-formula φ

$$\text{if } T \not\models \varphi \text{ then there exists } I \in [\![T]\!] \text{ such that } I \not\models \varphi \ .$$

The proof is by induction on the structure of the formula φ and under the assumption that $T \not\models \varphi$ we construct its implementation (i_0, I) such that $i_0 \not\models \varphi$. During the construction we will write that we add a transition $i_0 \xrightarrow{n} (i'_0, I')$ for an implementation (i'_0, I'), meaning that together with this transition we implicitly add also a disjoint copy of I' rooted at i'_0 to the implementation I.

The induction basis, where $\varphi = \mathsf{true}$ and $\varphi = \mathsf{false}$, is trivial.

- Case 1: $\varphi = \varphi_1 \wedge \varphi_2$. By the definition of $\models$ either $T \not\models \varphi_1$ or $T \not\models \varphi_2$ Assume w.l.o.g. that $T \not\models \varphi_1$. By applying the induction hypothesis there is $I \in [\![T]\!]$ such that $I \not\models \varphi_1$ and we conclude that $I \not\models \varphi_1 \wedge \varphi_2$.

- Case 2: $\varphi = \langle k \rangle \psi$. Assume that $T \not\models \langle k \rangle \psi$, which is the case if for all $t_0 \xrightarrow{\ell} t$ we have either (1) $[\![\ell]\!] \not\subseteq [\![k]\!]$ or (2) $(t, T) \not\models \psi$. We construct an implementation $(i_0, I) \in [\![T]\!]$ as follows. For every $t_0 \xrightarrow{\ell} t$ such that (1) is satisfied, we add the transition $i_0 \xrightarrow{n} (i'_0, I')$ into I where $n \in [\![\ell]\!] \setminus [\![k]\!]$ and $(i'_0, I') \leq_m (t, T)$ (the implementation (i'_0, I') exists by local consistency of T and well-formedness of the label-set). For every $t_0 \xrightarrow{\ell} t$ such that (2) is satisfied, we have by induction hypothesis an implementation $(i'_0, I') \in [\![(t, T)]\!]$ such that $i'_0 \not\models \psi$. We add $i_0 \xrightarrow{m} (i'_0, I')$ to I for some $m \in [\![\ell]\!]$. It is easy to see that $I \leq_m T$, and moreover $I \not\models \langle k \rangle \psi$ by the construction.

- Case 3: $\varphi = [k]\psi$. Assume that $T \not\models [k]\psi$. Then there exists $t_0 \overset{\ell}{\dashrightarrow} t$ such that $[\![\ell]\!] \cap [\![k]\!] \neq \emptyset$ and $(t, T) \not\models \psi$. By induction hypothesis there exists $(i'_0, I') \in [\![(t, T)]\!]$ such that $i'_0 \not\models \psi$. Let $(i_0, I) \in [\![T]\!]$ be some implementation of T (which exists by local consistency of T) where we add the transition $i_0 \xrightarrow{n} (i'_0, I')$ with $n \in [\![\ell]\!] \cap [\![k]\!]$. Clearly, we still have $I \leq_m T$ and moreover the transition $i_0 \xrightarrow{n} i'_0$ ensures that $i_0 \not\models [k]\psi$.

■

A similar completeness result in the setting of partial Kripke structures can be found also in [AH06].

We now study completeness with respect to modal refinement, that is the completeness definition considered in [Lar89]. In this article, it was shown that for classical modal transition systems (with the label-set K_{action} from Example 5), the LSHML logic is complete with respect to refinement. We first observe that the result does not extend to general LSMTSs.

Theorem 59. *The logic LSHML is incomplete with respect to modal refinement.*

Proof. Consider the systems S and T from Figure 6. By case analysis it is easy to verify that $s_0 \models \varphi$ if and only if $t_0 \models \varphi$ for any LSHML-formula φ. However, as argued before, $S \not\leq_m T$. ■

On the other hand, if we consider only deterministic systems, LSHML is complete even with disjunction, as proved below. We let $\mathcal{F}(S) = \{\varphi \mid S \models \varphi\}$ denote the set of all LSHML-formulae satisfied by S.

Theorem 60 (Completeness with respect to modal refinement for deterministic LSMTSs)**.** *Let $S, T \in \mathcal{M}_{(K,\sqsubseteq)}$ be deterministic LSMTSs (see Definition 24) and assume that the label refinement relation $\sqsubseteq$ is complete. Then*

$$\mathcal{F}(T) \subseteq \mathcal{F}(S) \quad \Longrightarrow \quad S \leq_m T \ .$$

Proof. Assume that $\mathcal{F}(T) \subseteq \mathcal{F}(S)$. We define a relation $R \subseteq S \times T$ by

$$R = \{(s,t) \mid \mathcal{F}((t,T)) \subseteq \mathcal{F}((s,S))\}.$$

We show that R is a relation witnessing $S \leq_m T$. Clearly $(s_0, t_0) \in R$ for the respective initial states. Let $(s,t) \in R$.

- First, assume that $s \overset{k}{\dashrightarrow} s'$. Clearly, $t \overset{\ell}{\dashrightarrow} t'$ for some ℓ such that $[\![k]\!] \cap [\![\ell]\!] \neq \emptyset$, otherwise the formula $[k]\mathsf{false}$ is satisfied in (t,T) but not in (s,S), contradicting our assumption that $\mathcal{F}((t,T)) \subseteq \mathcal{F}((s,S))$. By the determinism of T there can only be one such ℓ with $[\![k]\!] \cap [\![\ell]\!] \neq \emptyset$.

  For the sake of contradiction assume that $k \not\sqsubseteq \ell$. By completeness of label refinement we get $[\![k]\!] \not\subseteq [\![\ell]\!]$. Thus, there exists some $m \in [\![k]\!] \setminus [\![\ell]\!]$. The formula $[m]\mathsf{false}$ holds in (t,T) due to the choice of m and the absence of any other may transition having any common implementation labels with $[\![k]\!]$, hence in particular also with $[\![m]\!] = \{m\}$. However, $[m]\mathsf{false}$ does not hold in (s,S), contradicting the assumption that $\mathcal{F}((t,T)) \subseteq \mathcal{F}((s,S))$. Thus, we can assume the existence of $t \overset{\ell}{\dashrightarrow} t'$ with $k \sqsubseteq \ell$.

  Now we need to argue that $(s',t') \in R$. Assume that this is not the case. Then we have $\mathcal{F}((t',T)) \not\subseteq \mathcal{F}((s',S))$, and therefore there is a formula φ' such that $(t',T) \models \varphi'$ and $(s',S) \not\models \varphi'$. Consider the formula $\varphi = [k]\varphi'$. Again $(t,T) \models \varphi$, but $(s,S) \not\models \varphi$. This contradicts the assumption that $\mathcal{F}((t,T)) \subseteq \mathcal{F}((s,S))$. Thus $(s',t') \in R$.

- Second, assume that $t \xrightarrow{\ell} t'$. As in the previous item, there must be a transition $s \xrightarrow{k} s'$ such that $[\![k]\!] \subseteq [\![\ell]\!]$, otherwise the formula $\langle\ell\rangle\mathsf{true}$ is satisfied in (t,T) but not in (s,S), contradicting the assumption that $\mathcal{F}((t,T)) \subseteq \mathcal{F}((s,S))$. By the determinism of S we know that $s \xrightarrow{k} s'$ is a unique transition such that $[\![k]\!] \subseteq [\![\ell]\!]$ and due to the completeness of label refinement we know that $k \sqsubseteq \ell$.

  Remains to argue that $(s',t') \in R$. Assume that this is not the case. The arguments are similar to the previous case by considering the formula $\langle\ell\rangle\varphi'$ where $t' \models \varphi'$ and $s' \not\models \varphi'$. Thus $(s',t') \in R$ and this completes the proof.

■

5. Conclusion

We introduced label-structured modal transition systems, a basis for a specification formalism that generalizes modal theories such as weighted and multi-weighted modal transition systems. Our work contributes to the long-term objective towards the unification of existing specification theories through a common framework. A full specification theory for label-structured modal transition systems was developed, including the notions of modal and thorough refinement, consistency, determinization and deterministic-hull and a number of completeness results, often conditioned (as expected) by the requirement of determinism. We showed soundness results for the operators of parallel composition, conjunction and quotient. The specification theory was concluded by suggesting an extension of Hennessy-Milner logic to handle quantitative aspects and by showing the interplay between the logic and the refinement theory in a similar way as known from the classical theory of labelled transition systems and bisimulation.

Most of the proof techniques were generalizations of the techniques developed for concrete instances of the framework like modal transition systems, however, the general theorems provide novel results for particular instances like weighted and multi-weighted modal transition systems. Finally, we consider the uniform and complete presentation of the main aspects of the suggested specification theory as a contribution on its own.

There are a few instances of recently studied extensions of modal transition systems that cannot be captured in our framework. Here we list some of them as possible directions for future research.

Abstract Probabilistic Automata [DKL+11] and constraint Markov Chains [CDL+10] are recently introduced stochastic extensions of modal transition systems. One of the major difficulties is how to capture and generalize the satisfaction relation which is (unlike to our framework) based on a redistribution of weights from one to several transitions.

The theories based on the optimistic approach introduced in Interface Automata [dAH01, dAHS02] are also hard to capture by our framework. Here the semantics of a given specification is viewed as a two player game. The approach is optimistic in the sense that two specifications can be composed if and only if there exists at least one environment in which they can cooperate. The input and output modalities are orthogonal to may and must ones, which suggests that the label-structured modal transition systems need to be further extended to capture this phenomenon.

Finally, it would be of interest to consider models that manipulate data like in the spirit of sociable interfaces [dAdSF+05, AdAdS+06] and see if they can be described in the framework of label-structured modal transition systems.

E A Logic for Accumulated-Weight Reasoning on Multi-weighted Modal Automata

Sebastian Bauer
Institut für Informatik, Ludwig-Maximilians-Universität München, Germany

Line Juhl Kim G. Larsen
Aalborg University, Department of Computer Science, Denmark

Axel Legay
INRIA/IRISA, Rennes Cedex, France

Jiří Srba
Aalborg University, Department of Computer Science, Denmark

Abstract Multiweighted modal automata provide a specification theory for multi-weighted transition systems that have recently attracted interest in the context of energy games. We propose a simple fragment of CTL that is able to express properties about accumulated weights along maximal runs of multiweighted modal automata. Our logic is equipped with a game-based semantics and guarantees both soundness (formula satisfaction is propagated to the modal refinements) as well as completeness (formula non-satisfaction is propagated to at least one of its implementations). We augment our theory with a summary of decidability and complexity results of the generalized model checking problem, asking whether a specification—abstracting the whole set of its implementations—satisfies a given formula.

1. Introduction

Modal transition systems [LT88a] (MTS) have been recently studied as a suitable specification formalism in connection with step-wise design of component-based systems. This model is essentially a labelled transition system with two kinds of transition relations: a *may* (allowed) and *must* (compulsory) transition relation. During the system design process the must-transitions must be preserved, while the may-transitions may be omitted.

One of the key elements in model-based design is the notion of *refinement*. Specifications are gradually refined into more concrete ones until we arrive at the most concrete specification (called *implementation*) that cannot be refined any more. Taking the point of view that implementations can be seen as (abstractions of) the final products like executable systems we want to implement, we get the natural notion of so-called *thorough refinement*: specification S_1 thoroughly refines S_2 if any implementation of S_1 is also an implementation of S_2. Unfortunately, the thorough refinement preorder is computationally hard and deciding thorough refinement between two specifications is EXPTIME-complete [BKLS09a]. Hence, for feasibility reasons, thorough refinement is often approximated by *modal refinement* (defined directly on specifications in a bisimulation-like manner). Deciding modal refinement is possible in deterministic polynomial time [BKLS09b] and two specifications that are in modal refinement are also in thorough refinement, though not the other way round [LNW07a].

On the logical counter-part of the theory, already the first work on MTS [LT88a] introduced Hennessy-Milner logic for MTS with a model checking procedure on the specifications to decide whether all of its implementations satisfy the formula. Later, (3-valued) extensions of the problem were considered for several branching and linear time logics [HJS01, BG00, GHJ01, GC05, WGC09, GP09, BvK10]. In this paper we study the model checking problem for multiweighted modal automata. In comparison to other related works, we do not deal with Kripke MTS, but we consider multiweighted modal automata, that are finite MTS with vectors of integer intervals as labels. These (multi-)weighted automata have recently been in the focus of the research community but treated mainly as reachability/infinite runs problems with cost/energy objectives without any logical characterization. The theory of multiweighted modal automata constitutes an abstraction theory for multiweighted automata which we already studied in the context of energy games [FJLS11].

As a first contribution we investigate model checking formulae of a simple fragment of CTL with atomic propositions referring explicitly to *accumulated (multi-)weights* along maximal runs in the automata. This logic is inspired by the one we recently proposed in [JLS12], but with the crucial addition of a game semantics that is needed to prove completeness with respect to refinement. More precisely, the semantics of the logic is defined according to a game-based interpretation with two players, the *must-player* and the *may-player*. The intuition is that the selections of the must-player can be realized in *every* possible implementation, whereas the may-player can choose between all the design choices of the specification that are not yet fixed. The fact that our semantics

guarantees that *all* implementations satisfy a formula[1], immediately implies preservation of formula satisfaction under refinement. This leads to a new aspect not seen in previous works, namely that of *completeness* of our logic: if all implementations of a specification satisfy a formula, so does the specification. Completeness is also of practical interest as it allows for counterexample generation.

As our second contribution, we provide an overview of decidability and complexity analysis of the model checking problems for our proposed accumulated-weight logic, showing that in their full generality the problems are undecidable but by imposing some natural restrictions we get decidability and the problems in many cases specialize to the well studied problems in the theory. Throughout the paper, we restrict ourselves to the four EF, EG, AF and AG fragments of CTL as they provide a good balance between the expressiveness of the logic for practical applications and on the other hand allow us to conclude at least partial decidability results.

Related work: Weighted models have been widely studied over the past years [DG07, DKV09]. Other works on quantitative models with quantitative reasoning include [DM10, BG09, KS08]. The main difference with our model is that they do not consider formalisms capable of a step-wise refinement process like we do through the modalities.

In a very recent paper [BCHK11], the branching time logic CTL has been extended to quantitative objectives that allow reasoning on several accumulated weights. The underlying model used in this work is the one of quantitative Kripke structures and the paper presents a largest decidable fragment of this logic. The logic we propose is only a fragment of CTL extended with accumulative reasoning on multiweights, however, we use multiweighted *modal* transition systems as the underlying specification model, not just the implementations. Hence, contrary to others, our model is able to express both allowed and required behaviours, and also looseness of the quantitative information, not possible in [BCHK11].

2. Multiweighted Modal Automata

We define $[a, b] = \{n \in \mathbb{Z} \mid a \leq n \leq b\}$ for $a \leq b$, $a \in \mathbb{Z} \cup \{-\infty\}$, $b \in \mathbb{Z} \cup \{\infty\}$ to denote the interval with lower bound a and upper bound b, and we use $\mathbb{W}$ to denote the set of all such intervals. A *k-weight interval*, for a natural number $k \geq 1$, is an element $\overline{W} \in \mathbb{W}^k$, in other words a vector consisting of k intervals. Projection on the i-th interval, $1 \leq i \leq k$, is denoted by $\overline{W}[i]$. Moreover, we write $\overline{W} \subseteq \overline{V}$ where $\overline{W}, \overline{V} \in \mathbb{W}^k$ iff $\overline{W}[i] \subseteq \overline{V}[i]$ for all i, $1 \leq i \leq k$. The set of all singleton intervals of the form $[a, a]$ is denoted by $\mathbb{W}_1$ and by misusing the notation the set $\mathbb{W}_1^k$ will often be identified with $\mathbb{Z}^k$ and we refer to its elements as *k-weights* or just *weights*. The addition of two weights $\overline{w}_1, \overline{w}_2 \in \mathbb{Z}^k$ is defined by $(\overline{w}_1 \oplus \overline{w}_2)[i] = \overline{w}_1[i] + \overline{w}_2[i]$.

A k-weighted modal automaton is a variant of the classical modal transition systems [LT88a] where transitions are labeled by k-weight intervals (instead of actions). A k-weighted modal automaton differs from the usual weighted automaton by distinguishing two transition relations: the *may*-transition relation expresses which transitions are

[1]Some other works seek for the existence of one such implementation [WGC09].

optional for any implementation, and the *must*-transition relation contains those transitions that are mandatory for any implementation.

Definition 1. A *k-weighted modal automaton* is a tuple $\mathcal{S} = (S, s_0, \overline{w}_0, \dashrightarrow, \longrightarrow)$ where S is a set of states, $s_0 \in S$ is an initial state, $\overline{w}_0 \in \mathbb{Z}^k$ is an initial weight, and $\longrightarrow \subseteq \dashrightarrow \subseteq S \times \mathbb{W}^k \times S$ are the must- and may-transition relations, respectively.

The set of all k-weighted modal automata is denoted by $\mathbb{M}$. Given a k-weighted modal automaton $\mathcal{S} = (S, s_0, \overline{w}_0, \dashrightarrow, \longrightarrow) \in \mathbb{M}$, a state $s \in S$ and a weight $\overline{w} \in \mathbb{Z}^k$, we write $\mathcal{S}_{(s,\overline{w})}$ for the k-weighted modal automaton $(S, s, \overline{w}, \dashrightarrow, \longrightarrow)$ where the initial state s_0 and the initial weight $\overline{w}_0$ are replaced by s and $\overline{w}$, respectively.

We note that although we have omitted actions from the definition, this is not a real restriction: given a finite set of actions $\Sigma = \{a_1, \ldots, a_n\}$ and a k-weighted modal automaton, we can encode the actions in a $(k+n)$-weighted modal automaton by introducing for every i-th action $a_i \in \Sigma$, $1 \leq i \leq n$, a new weight coordinate $k+i$ which equals 1 iff the transition is labelled by action a, and 0 otherwise.

A k-weighted modal automaton $\mathcal{S} = (S, s_0, \overline{w}_0, \dashrightarrow, \longrightarrow)$ is an *implementation* iff all labels are singleton intervals from $\mathbb{W}_1^k$ and $\dashrightarrow = \longrightarrow$. In other words, all allowed transitions are also implemented and all choices of concrete weights from the interval are realized.

We shall now introduce the classical notion of modal refinement [LT88a] extended with the interval refinement, defined similarly as in [JLS12].

Definition 2. Let $\mathcal{S}_1 = (S_1, s_{0,1}, \overline{w}_{0,1}, \dashrightarrow_1, \longrightarrow_1)$ and $\mathcal{S}_2 = (S_2, s_{0,2}, \overline{w}_{0,2}, \dashrightarrow_2, \longrightarrow_2)$ be two k-weighted modal automata. We say that $\mathcal{S}_1$ *modally refines* $\mathcal{S}_2$, written as $\mathcal{S}_1 \leq_m \mathcal{S}_2$, if $\overline{w}_{0,1} = \overline{w}_{0,2}$ and there is a binary relation $R \subseteq S_1 \times S_2$ such that $(s_{0,1}, s_{0,2}) \in R$ and for all $(s_1, s_2) \in R$:

1. whenever $s_1 \overset{\overline{W}_1}{\dashrightarrow}_1 s_1'$ then there exists $s_2 \overset{\overline{W}_2}{\dashrightarrow}_2 s_2'$ such that $\overline{W}_1 \subseteq \overline{W}_2$ and $(s_1', s_2') \in R$,
2. whenever $s_2 \overset{\overline{W}_2}{\longrightarrow}_2 s_2'$ then there exists $s_1 \overset{\overline{W}_1}{\longrightarrow}_1 s_1'$ such that $\overline{W}_1 \subseteq \overline{W}_2$ and $(s_1', s_2') \in R$.

Clearly, modal refinement is a preorder. The set of all implementations of a modal automaton $\mathcal{S}$ is then defined by $[\![\mathcal{S}]\!]_{\text{impl}} = \{\mathcal{I} \mid \mathcal{I} \leq_m \mathcal{S} \text{ and } \mathcal{I} \text{ is an implementation }\}$.

Example 3. Examples of 2-weighted modal automata are drawn in Figure 1. Figure 1b shows a specification $\mathcal{S}$ of a Mars vehicle. The vehicle has a battery and a container for carrying rocks that it collects. The first weight denotes changes in the battery level, while the second weight denotes changes in the accumulated volume of rocks. In s_0 the battery can be charged, while state s_1 enables the search for new rocks or the deposit of a rock. The abilities to collect a big rock and to reset are not required behaviours in a possible implementation. The automaton $\mathcal{T}$ given in Figure 1a is a refinement of $\mathcal{S}$, demonstrated by the refinement relation $\{(t_0, s_0), (t_1, s_1), (t_2, s_2), (t_3, s_3)\}$. This refinement is furthermore an implementation of $\mathcal{S}$ as it implements two of the three

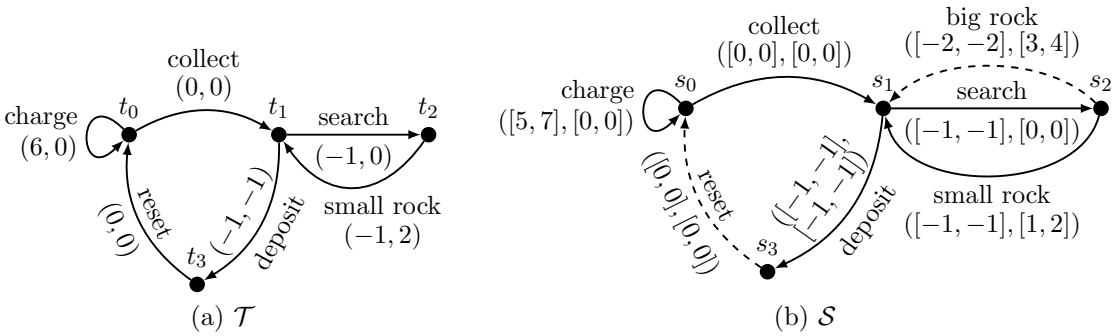


Figure 1.: Examples of 2-weighted modal automata

proper may-transitions present in $\mathcal{S}$ (and all must-transitions) and has all transitions labelled with k-weights from the corresponding interval in $\mathcal{S}$.

3. Games on Multiweighted Modal Automata and Logic $\mathcal{L}$

As motivated in the introduction, the semantics of our logic will be based on a game characterization in order to be able to argue for its completeness. We shall now introduce this game.

Let $\mathcal{S} = (S, s_0, \overline{w}_0, \dashrightarrow, \longrightarrow)$ be a k-weighted modal automaton. The game is played in rounds, with two players called the *must-* and *may-player*. The set of states S is partitioned into *must-* and *may-states* in which it is the turn of the must- and may-player, respectively. A *must-state* is a state in which there is at least one outgoing must-transition, otherwise the state is a *may-state*. *Configurations* are of the form $(s, \overline{w})$ where $s \in S$ and $\overline{w} \in \mathbb{Z}^k$ is the so far accumulated weight. Each round starts from the current configuration $(s, \overline{w})$, initially $(s_0, \overline{w}_0)$, and has two steps: (i) selection of a transition and (ii) choosing a specific weight from the weight interval of the chosen transition. Formally,

(i) a) If s is a must-state, then the must-player chooses some must-transition $(s, \overline{W}, s') \in \longrightarrow$.

b) If s is a may-state, then the may-player chooses some may-transition $(s, \overline{W}, s') \in \dashrightarrow$ or decides to stop the game.

(ii) Afterwards, if the game was not stopped and a transition $(s, \overline{W}, s')$ was selected, the may-player chooses a weight $\overline{w}' \in \overline{W}$.

The pair $(s', \overline{w} \oplus \overline{w}')$ now becomes the new current configuration and the game continues with a next round.

The intuition is that the must-player can only select must-transitions in must-states and thus the selections (or moves) of the must-player can be realized in *every* possible implementation. The may-player can choose between all the design choices of $\mathcal{S}$ which

are not yet fixed, i.e. the may-player can choose whether to take any may-transition (and in this case, which one) or not, and which weight to pick from a weight interval (both for may- as well as must-transitions).

Any maximal sequence (finite that cannot be prolonged using a must-transition or infinite) of configurations $(s_0, \overline{w}_0)(s_1, \overline{w}_1)\dots$ such that for all $i \geq 0$, we have $s_i \overset{\overline{W}_i}{\dashrightarrow} s_{i+1}$ and $\overline{w}_{i+1} = \overline{w}_i \oplus \overline{v}_i$ where $\overline{v}_i \in \overline{W}_i$, is called a *run* on $\mathcal{S}$. Let $runs(s, \overline{w})$ denote the set of all runs starting from $(s, \overline{w})$ on $\mathcal{S}_{(s,\overline{w})}$. Any run complying with the above rules (i) and (ii) is called a *play* in $\mathcal{S}$. A *strategy of the must-player* is a function σ that maps every finite prefix $(s_0, \overline{w}_0)(s_1, \overline{w}_1)\dots(s_r, \overline{w}_r)$ of a play, ending in a must-state s_r, to a must-transition $(s_r, \overline{W}, s') \in \longrightarrow$. For a fixed strategy σ of the must-player, we define the set $plays(\sigma, (s, \overline{w}))$ of all plays starting from $(s, \overline{w})$ on $\mathcal{S}_{(s,\overline{w})}$ in which the choice of the next must-transition is according to σ. For a run $\gamma \in runs(s, \overline{w})$ the projection to the i-th configuration is denoted by γ_i.

We are now ready to define a fragment $\mathcal{L}$ of the logic CTL to express properties about accumulated weights of maximal runs in multiweighted modal automata. The satisfaction relation will be defined via the games introduced above and we will show that the logic is sound and complete w.r.t. refinement. In what follows let k implicitly represent the number of weight coordinates.

The set of *linear expressions* is given by the abstract syntax $e ::= \langle i \rangle \cdot c \mid e + e$ where $1 \leq i \leq k$ and $c \in \mathbb{Z}$. The *$\mathcal{L}$-formulae* are given by the abstract syntax

$$\varphi, \varphi_1, \varphi_2 ::= e \bowtie b \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2$$
$$\psi ::= \mathsf{AG}\varphi \mid \mathsf{AF}\varphi \mid \mathsf{EG}\varphi \mid \mathsf{EF}\varphi$$

where e is a linear expression, $\bowtie \in \{<, \leq, =, \neq, \geq, >\}$, $b \in \mathbb{Z} \cup \{-\infty, \infty\}$.

In order to give the semantics, we first define, for a linear expression e and a weight $\overline{w} \in \mathbb{Z}^k$, its denotational semantics $[\![e]\!]_{\overline{w}} \in \mathbb{Z}$ by $[\![\langle i \rangle \cdot c]\!]_{\overline{w}} = \overline{w}[i] \cdot c$ and $[\![e_1 + e_2]\!]_{\overline{w}} = [\![e_1]\!]_{\overline{w}} + [\![e_2]\!]_{\overline{w}}$. The satisfaction of an $\mathcal{L}$-formula ψ in a configuration $(s, \overline{w})$ is now given in Figure 2. Notice that the semantics of the AF and AG fragments can also be defined using strategies. However, as these reduce essentially to one-player games, we prefer to give their direct definitions for clarity.

By the EF, AF, EG and AG fragments we refer to the four main fragments of $\mathcal{L}$, namely formulae of the form $\mathsf{EF}\varphi$, $\mathsf{AF}\varphi$, $\mathsf{EG}\varphi$ or $\mathsf{AG}\varphi$, respectively. If φ moreover does not contain any disjunction (conjunction), we call the fragment *disjunction-free* (*conjunction-free*). In the following, for $\mathcal{S} = (S, s_0, \overline{w}_0, \dashrightarrow, \longrightarrow) \in \mathbb{M}$, we shortly write $\mathcal{S} \models \psi$ whenever $(s_0, \overline{w}_0) \models \psi$.

Example 4. As an example of an $\mathcal{L}$-formula consider $\psi = \mathsf{EG}\big(\langle 1 \rangle \geq 0 \wedge \langle 1 \rangle \leq 10 \wedge \langle 2 \rangle \geq 0 \wedge \langle 2 \rangle \leq 6\big)$ in connection with the modal automata from Figure 1. The formula claims the existence of a strategy for the must-player such that the battery level is between 0 and 10 and the volume of accumulated rocks is between 0 and 6 in any configuration. By consulting Figure 1 we can see that $\mathcal{S} \models \psi$. No matter what weight the may-player chooses within the intervals and regardless whether the may-player chooses to stop or not in s_3, the must-player can always keep the two accumulated weights between the

$$
\begin{aligned}
(s,\overline{w}) &\models e \bowtie b && \text{iff} && [\![e]\!]_{\overline{w}} \bowtie b \\
(s,\overline{w}) &\models \varphi_1 \wedge \varphi_2 && \text{iff} && (s,\overline{w}) \models \varphi_1 \text{ and } (s,\overline{w}) \models \varphi_2 \\
(s,\overline{w}) &\models \varphi_1 \vee \varphi_2 && \text{iff} && (s,\overline{w}) \models \varphi_1 \text{ or } (s,\overline{w}) \models \varphi_2 \\
(s,\overline{w}) &\models \mathsf{AG}\varphi && \text{iff} && \forall \gamma \in \mathit{runs}(s,\overline{w})\ \forall i : \gamma_i \models \varphi \\
(s,\overline{w}) &\models \mathsf{AF}\varphi && \text{iff} && \forall \gamma \in \mathit{runs}(s,\overline{w})\ \exists i : \gamma_i \models \varphi \\
(s,\overline{w}) &\models \mathsf{EG}\varphi && \text{iff} && \text{there exists a strategy } \sigma \text{ of the must-player such that} \\
& && && \forall \gamma \in \mathit{plays}(\sigma,(s,\overline{w}))\ \forall i : \gamma_i \models \varphi \\
(s,\overline{w}) &\models \mathsf{EF}\varphi && \text{iff} && \text{there exists a strategy } \sigma \text{ of the must-player such that} \\
& && && \forall \gamma \in \mathit{plays}(\sigma,(s,\overline{w}))\ \exists i : \gamma_i \models \varphi
\end{aligned}
$$

Figure 2.: Semantics of the logic $\mathcal{L}$

In s_0:

if $\langle 1\rangle = 0 \ \vee\ (\langle 1\rangle = 1 \wedge \langle 2\rangle = 0)$ **then** charge
else collect

In s_1:

if $(\langle 1\rangle \geq 4 \wedge \langle 2\rangle = 0) \ \vee\ (\langle 1\rangle = 3 \wedge \langle 2\rangle \in \{0,1\})$ **then** search
else deposit

Figure 3.: A strategy for the must-player

bounds. A sufficient strategy for the must-player is seen in Figure 3. Notice that the choice for the must-player in the state s_2 is always to go for a small rock, thus this is omitted in the strategy. As we will see shortly, the logic $\mathcal{L}$ is sound, implying that an $\mathcal{L}$-formula satisfied by $\mathcal{S}$ is also satisfied by all refinements of $\mathcal{S}$, thus $\mathcal{T} \models \psi$ as well.

On the other hand, the formula $\psi' = \mathsf{AG}\big(\langle 1\rangle + \langle 2\rangle \leq 20\big)$ can be easily seen *not* to be satisfied by $\mathcal{S}$. Since the logic $\mathcal{L}$ in this paper is proved complete, it allows us to generate counter-examples: implementations of $\mathcal{S}$ that do not satisfy the formula ψ'. Indeed, the implementation $\mathcal{T}$ does not satisfy ψ' as a run may consist of continuously charging using the selfloop in t_0.

Remark 1. Note that the formula $\mathsf{EF}\psi$ cannot in general be expressed as the negation of $\mathsf{AG}\neg\psi$ as opposed to the classical CTL. Consider for example a 1-weighted modal automaton with just one transition $s_0 \overset{[2,2]}{\dashrightarrow} s_1$. Clearly, $(s_0,\overline{0}) \not\models \mathsf{EF}\ \langle 1\rangle = 2$ while also $(s_0,\overline{0}) \not\models \mathsf{AG}\ \langle 1\rangle \neq 2$. Similarly, AF is not dual with EG. On the other hand, as expected, the classical duality laws for EF and AG, as well as AF and EG, hold on implementations.

Now we can show that satisfaction of any $\mathcal{L}$-formula is preserved by modal refinement. In particular this means that if $\mathcal{S} \models \psi$ then any implementation of $\mathcal{S}$ also satisfies ψ.

Theorem 5. *Let $\mathcal{S} \in \mathbb{M}$ be a k-weighted modal automaton and ψ be an $\mathcal{L}$-formula. Then*

$$\mathcal{S} \models \psi \quad \Longrightarrow \quad \left(\forall\, \mathcal{T} \in \mathbb{M} : \mathcal{T} \leq_m \mathcal{S} \implies \mathcal{T} \models \psi\right) .$$

Proof. It is clear that the theorem holds for any formula φ which is a logical combination of atomic propositions, because it only refers to the current accumulated weight (which is $\overline{0}$ in the initial configuration). Let φ be a logical combination of atomic propositions, and let ψ be a formula of the form $\mathsf{EF}\varphi$, $\mathsf{EG}\varphi$, $\mathsf{AF}\varphi$, $\mathsf{AG}\varphi$. Let $\mathcal{T} \in \mathbb{M}$ be a k-weighted modal automaton such that $\mathcal{T} \leq_m \mathcal{S}$.

For the case $\psi \in \{\mathsf{AG}\varphi, \mathsf{AF}\varphi\}$, consider a run $\gamma_\mathcal{T}$ in $\mathcal{T}$. It is clear that from the modal refinement $\mathcal{T} \leq_m \mathcal{S}$ it follows that there exists a run $\gamma_\mathcal{S}$ in $\mathcal{S}$ such that both runs have the same length, and for all i, the state of $(\gamma_\mathcal{T})_i$ is in a modal refinement relation with $(\gamma_\mathcal{S})_i$, i.e. $\mathcal{T}_{(\gamma_\mathcal{T})_i} \leq_m \mathcal{S}_{(\gamma_\mathcal{S})_i}$, and the accumulated weights of $(\gamma_\mathcal{T})_i$ and $(\gamma_\mathcal{S})_i$ coincide. Hence $\mathcal{T} \models \psi$.

Consider now the case $\psi \in \{\mathsf{EG}\varphi, \mathsf{EF}\varphi\}$. By assumption we know that there is a strategy σ for the must-player that witnesses $\mathcal{S} \models \psi$. We can iteratively construct a strategy σ' for the must-player on $\mathcal{T}$ as follows. Assume a configuration $(t, \overline{w})$ with a must-state t that is related (the states are related by modal refinement and the weights are the same) to a configuration $(s, \overline{w})$ of $\mathcal{S}$ such that there is a strategy for the must-player witnessing $\mathcal{S}_{(s,\overline{w})} \models \psi$. So for $(s, \overline{w})$ there exists a choice of the next must-transition such that for any choice of the weight of the may-player, in the next configuration there is again a strategy for the must-player which witnesses the satisfaction of ψ. This must-transition is also present in $\mathcal{T}$, due to modal refinement, hence the choice of the must-transition by σ can be simulated by the strategy σ' in $\mathcal{T}$. Finally, note that the may-player in $\mathcal{T}$ has at most the choices (for weights and may-transitions) as there are in $\mathcal{S}$. Hence $\mathcal{T} \models \psi$. ■

We shall now argue for the completeness of our logic. The proof is more straightforward for the formulae $\mathsf{AG}\varphi$ and $\mathsf{AF}\varphi$, and for the formulae $\mathsf{EG}\varphi$ and $\mathsf{EF}\varphi$ the proof relies on the fact that in our turn-based games the absence of a winning strategy implies the existence of a spoiling strategy for the opponent.

Theorem 6. *Let $\mathcal{S} \in \mathbb{M}$ be a k-weighted modal automaton and ψ be an $\mathcal{L}$-formula. Then*

$$\left(\forall \mathcal{I} \in [\![\mathcal{S}]\!]_{\text{impl}} : \mathcal{I} \models \psi\right) \quad \Longrightarrow \quad \mathcal{S} \models \psi \ .$$

Proof (sketch). Let $\mathcal{S} = (S, s_0, \overline{w}_0, \dashrightarrow, \longrightarrow) \in \mathbb{M}$ be a k-weighted modal automaton. Let φ be a logical composition of atomic propositions of the form $e \bowtie b$. Since any k-weighted modal automaton has an implementation $\mathcal{I}$ and $\mathcal{I} \models \varphi$ by assumption, necessarily also $\mathcal{S} \models \varphi$.

Now consider the case $\psi \in \{\mathsf{AG}\varphi, \mathsf{AF}\varphi\}$. Let $\gamma_\mathcal{S}$ be a run in $\mathcal{S}$. Then, clearly, there exists an implementation $\mathcal{I} \in [\![\mathcal{S}]\!]_{\text{impl}}$ and a run $\gamma_\mathcal{I}$ in $\mathcal{I}$ such that both runs have the same length, and for all i, the state of $(\gamma_\mathcal{I})_i$ is in a modal refinement relation with $(\gamma_\mathcal{S})_i$, i.e. $\mathcal{I}_{(\gamma_\mathcal{I})_i} \leq_m \mathcal{S}_{(\gamma_\mathcal{S})_i}$, and the accumulated weights of $(\gamma_\mathcal{I})_i$ and $(\gamma_\mathcal{S})_i$ coincide. So essentially both runs have the same sequence of weights, and we can conclude that $\mathcal{S} \models \psi$.

Let us now consider the case $\psi \in \{\mathsf{EG}\varphi, \mathsf{EF}\varphi\}$. Assume that $\mathcal{S}$ does *not* satisfy ψ, i.e. $\mathcal{S} \not\models \psi$, then there does not exist any strategy σ for the must-player such that all plays $\gamma \in \mathit{plays}(\sigma, (s_0, \overline{w}_0))$ satisfy the respective property. We can infer the existence of a spoiling strategy of the may-player as follows. First, we define $\mathit{win} = \{(s, \overline{w}) \mid (s, \overline{w})$ is a configuration in $\mathcal{S}$ and must-player has a strategy in $\mathcal{S}_{(s,\overline{w})}$ witnessing $\psi\}$. Given a configuration $(s, \overline{w})$ in $\mathcal{S}$ such that $(s, \overline{w}) \notin \mathit{win}$ and s is a must-state, then for all choices of the must-player we know that there is a next choice of the may-player (selection of a weight from the interval) such that the next configuration is not in win. Similarly, if s is a may-state, we also know that there exists at least one choice of the next (may-)transition and weight such that the next configuration is not in win; otherwise it would contradict with $(s, \overline{w}) \notin \mathit{win}$.

Obviously, we can construct an implementation $\mathcal{I}$ of $\mathcal{S}$, in the form of a tree, for which the weights and the presence of transitions are chosen according to the above choices of the must- and may-players. It is clear that the must-player does not have any strategy witnessing the formula ψ since any play in $\mathcal{I}$ does not satisfy the respective property. This contradicts our assumption that all implementations satisfy ψ. Hence $\mathcal{S} \models \psi$. ■

4. Decidability and Complexity of the Logic $\mathcal{L}$

We shall now study the *model checking problem* for $\mathcal{L}$: given a finite k-weighted modal automaton $\mathcal{T}$ and an $\mathcal{L}$-formula ψ, the question is to decide whether $\mathcal{T} \models \psi$. As we will show, the problem is undecidable in general. Fortunately, there are several practically usable fragments of the logic for which we show decidability. In the rest of this section we shall implicitly assume that all input modal automata are finite. The first part concentrates on undecidability results, while the two following parts study the decidable fragments of the logic.

4.1. Undecidability of EF, EG and AF

We start by showing that in general the model checking problem is undecidable for three (EF, EG, AF) of the four fragments of the logic.

The authors of [BCHK11] propose a CTL logic on Kripke structures that can reason about multiple accumulated weights and they show undecidability of an unnested EG formula. As our semantics of $\mathcal{L}$ corresponds to normal CTL semantics when interpreted on implementations (the may-player has no choices, thus the existence of a strategy corresponds to the existence of a run), the undecidable formula constructed in [BCHK11] can be expressed using an EG formula from our logic $\mathcal{L}$. Since the operators are dual on implementations (see Remark 1), the model checking problems of the EG and AF fragments of $\mathcal{L}$ are undecidable, even for implementations.

Theorem 7 ([BCHK11])**.** *The model checking problems of the* EG *and* AF *fragments are undecidable, even for implementations.*

For the EF fragment we will describe a reduction from the halting problem of a 2-counter Minsky machine to the model checking problem of the EF fragment of the logic. Recall that a *Minsky machine* [Min67] consists of a finite number of instructions and two nonnegative integer counters initially both set to 0. Each instruction either increases one of the counters (an *increment* instruction) or tests for zero and decreases a counter (a *test-and-decrement* instruction). We say that a Minsky machine halts if it is possible to reach the last instruction called *halt* when starting from the first instruction. Otherwise it *loops*. It is well-known that the halting problem for Minsky machines is undecidable [Min67].

We now describe the reduction for the EF fragment where the problem becomes undecidable even if we restrict ourselves only to specifications where the may- and must-transitions coincide (though intervals are not necessarily singletons).

Let $1 : \mathsf{inst}_1;\ 2 : \mathsf{inst}_2;\ \ldots;\ n-1 : \mathsf{inst}_{n-1};\ n : \mathsf{halt}$ be a Minsky machine over the nonnegative integer counters c_1 and c_2. We construct a 9-weighted modal automaton $\mathcal{S} = (S, s_1, \overline{0}, \dashrightarrow, \longrightarrow) \in \mathbb{M}$ where every may-transition is also a must-transition and an $\mathcal{L}$-formulae $\mathsf{EF}\varphi$ such that $(s_1, \overline{0}) \models \mathsf{EF}\varphi$ iff the Minsky machine halts. The intuition behind the coordinates is as follows: $\langle 1\rangle$ represents the first counter, if $\langle 3\rangle$ is set to 1 then the may-player is testing if the first counter is empty, if $\langle 5\rangle$ is set to 1 then the may-player is testing if the first counter is nonempty, if $\langle 7\rangle$ is set to 1 then the may-player indicates that an increment instruction is not allowed. The role of the coordinates $\langle 2\rangle$, $\langle 4\rangle$, $\langle 6\rangle$ and $\langle 8\rangle$ is dual and corresponds to the second counter. Finally, if $\langle 9\rangle$ is nonzero, the halt instruction has been reached.

Let $S = \{s_i \mid 1 \le i \le n\}$ be the set of states. The transitions are of the following types, depending on the instructions of the Minsky machine (here $1 \le j \le 2$, $1 \le i, k, \ell \le n$).

1. For each instruction i: c_j `:=` c_j `+ 1; goto` k, we add the transitions
   - $s_i \xrightarrow{(1,0,0,0,0,0,[0,1],0,0)} s_k$ if $j = 1$, and
   - $s_i \xrightarrow{(0,1,0,0,0,0,0,[0,1],0)} s_k$ if $j = 2$.
2. For each instruction i: `if` c_j `= 0 then goto` k `else (`c_j `:=` c_j `- 1; goto` ℓ`)`, we add the transitions
   - $s_i \xrightarrow{(0,0,[0,1],0,0,0,0,0,0)} s_k$ and $s_i \xrightarrow{(-1,0,0,0,[0,1],0,0,0,0)} s_\ell$ if $j = 1$, and
   - $s_i \xrightarrow{(0,0,0,[0,1],0,0,0,0,0)} s_k$ and $s_i \xrightarrow{(0,-1,0,0,0,[0,1],0,0,0)} s_\ell$ if $j = 2$.
3. Finally, we add the transition $s_n \xrightarrow{(0,0,0,0,0,0,0,0,1)} s_n$.

Let now

$$\varphi_1 = \langle 1\rangle = 0 \ \wedge\ \langle 3\rangle = 1 \ \wedge\ \langle 5\rangle = 0 \qquad \varphi_2 = \langle 2\rangle = 0 \ \wedge\ \langle 4\rangle = 1 \ \wedge\ \langle 6\rangle = 0$$
$$\varphi_3 = \langle 1\rangle \ge 0 \ \wedge\ \langle 5\rangle = 1 \ \wedge\ \langle 7\rangle = 0 \qquad \varphi_4 = \langle 2\rangle \ge 0 \ \wedge\ \langle 6\rangle = 1 \ \wedge\ \langle 8\rangle = 0$$
$$\varphi_5 = \langle 5\rangle = 0 \ \wedge\ \langle 7\rangle = 1 \qquad \varphi_6 = \langle 6\rangle = 0 \ \wedge\ \langle 8\rangle = 1$$
$$\varphi_7 = \langle 9\rangle = 1 \ \wedge\ \langle 3\rangle = 0 \ \wedge\ \langle 4\rangle = 0 \ \wedge\ \langle 5\rangle = 0 \ \wedge\ \langle 6\rangle = 0 \ \wedge\ \langle 7\rangle = 0 \ \wedge\ \langle 8\rangle = 0.$$

and

$$\varphi = \varphi_1 \ \vee \ \varphi_2 \ \vee \ \varphi_3 \ \vee \ \varphi_4 \ \vee \ \varphi_5 \ \vee \ \varphi_6 \ \vee \ \varphi_7.$$

We now argue that $(s_1, \overline{0}) \models \mathsf{EF}\varphi$ iff the Minsky machine halts. Assume first that the machine halts. Then the must-player can reach the state s_n with some accumulated weight $\overline{w}$ by faithfully simulating the Minsky machine. If the may-player picks 0 in all intervals, φ_7 is satisfied in $(s_n, \overline{w})$, and thus also φ. If, on the other hand, the may-player picks 1 in any interval, this will force φ_1, φ_2, φ_3, φ_4, φ_5 or φ_6 to be true. Therefore $(s_0, \overline{0}) \models \mathsf{EF}\varphi$ holds regardless of the choices of the may-player.

Assume now that the Minsky machine does not halt. If the must-player does not cheat, s_n can never be reached and φ_7 can thus never be true. If the may-player chooses 0 in every interval, neither φ_1, φ_2, φ_3, φ_4, φ_5 nor φ_6 can be true in any configuration, hence $(s_0, \overline{0}) \models \mathsf{EF}\varphi$ can never be true.

We investigate what happens if the must-player cheats. This is possible either by (1) taking the transition $s_i \xrightarrow{(0,0,[0,1],0,0,0,0,0,0)} s_k$ while $c_1 > 0$ or (2) taking $s_i \xrightarrow{(-1,0,0,0,[0,1],0,0,0,0)} s_\ell$ while $c_1 = 0$ (and similarly for c_2).

In case (1), the may-player sets $\langle 3 \rangle = 1$. Since the accumulated weight in coordinates 3-8 can never be lowered, φ_1 can still hope to be true at some point. This would be possible if $\langle 1 \rangle = 0$, but for this not to happen the may-player sets $\langle 5 \rangle = 1$ should the other player try to decrement $\langle 1 \rangle$. This ensures that φ_1 cannot be true, so $(s_0, \overline{0}) \models \mathsf{EF}\varphi$ can never be true if the may-player chooses 0 in the remaining intervals (unless the must-player cheats in the other counter; here the may-player behaves analogously).

In case (2), the may-player sets $\langle 5 \rangle = 1$. Now φ_3 can still be true at some point. However, that would require $\langle 1 \rangle = 0$, and when incrementing $\langle 1 \rangle$, the may-player can set $\langle 7 \rangle = 1$, making sure that $\mathsf{EF}\varphi$ cannot become true. Similarly for c_2.

The above construction leads to the following theorem.

Theorem 8. *The model checking problem of the* EF *fragment is undecidable, even for specifications with intervals but where the may- and must-transition relations coincide.*

Notice that in the proof of Theorem 8 we use intervals to argue for undecidability. We will see that this is exactly the issue causing undecidability.

4.2. Decidability with Restricted Fragments

In order to obtain decidability results, we first restrict our model to contain only singleton intervals and notice that the EF fragment becomes decidable. Notice that this restriction is not sufficient for the two remaining undecidable fragments, EG and AF, since these are undecidable even for implementations.

Notice that a formula $\mathsf{EF}\varphi$ is true on a k-weighted modal automaton iff it is true on its *must-projection* that is obtained by removing all the may-transitions that do not have a corresponding must-transition. The reason is that the may-player can, in any may-state, decide to stop the game and hence the formula φ must be satisfied in some configuration reachable via must-transitions only. So in the following we assume that $\dashrightarrow = \longrightarrow$. Since the intervals are assumed singletons, we therefore need to consider only

implementations. The EF fragment was shown to be decidable for implementations in [BCHK11] (even if we allow nesting). We thus have the following decidability result.

Theorem 9 ([BCHK11])**.** *The model checking problem of the* EF *fragment of* $\mathcal{L}$ *against weighted modal automata with singleton intervals is decidable.*

The following alternative proof of the above theorem is not included in the submitted paper, as the theorem follows from [BCHK11], but for completeness we include this direct and self-contained proof. The proof is by reduction to the reachability problem on Petri nets. As expected, we store the accumulated weights in places of the Petri net. The main complication is that we need to remember also negative weights, something not possible in Petri nets. For that reason, for each coordinate we create two places p^+ and p^- such that tokens can be only added to them and the actual weight is represented by the difference between the number of tokens in p^+ minus the number of tokens in p^-. The checking of arbitrary boolean formula over simple linear expressions is then encoded inductively into Petri net fragments.

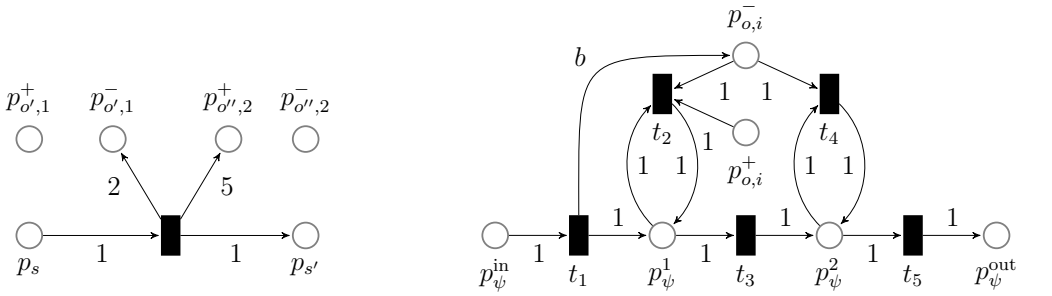
A *Petri net* is formally defined as a triple $N = (P, T, W)$, where P and T are finite sets of places and transitions, respectively, and $W : (P \times T) \cup (T \times P) \to \mathbb{N}$ is a weight function assigning a nonnegative integer to each arc. A *marking* is a function $M : P \to \mathbb{N}$ returning the number of tokens present in each place. Giving a marking M, any transition $t \in T$ may fire if $M(p) \geq W(p,t)$ for all $p \in P$. When a transition fires, it produces a marking M' obtained as $M'(p) = M(p) - W(p,t) + W(t,p)$ for all $p \in P$. A marking M is *reachable* in N from an initial marking M_0 if there exists a series of transition firings ending in the marking M.

Proof of Theorem 9. We show a polynomial time reduction to the decidable reachability problem on Petri nets. Let $\mathcal{S} = (S, s_0, \overline{w}_0, \dashrightarrow, \longrightarrow)$ be a k-weighted modal automaton with singleton intervals. We construct a Petri net $N = (P, T, W)$ and two markings M_0 and M_f such that $\mathcal{S} \models \mathsf{EF}\varphi$ iff M_f is reachable in N with the initial marking M_0.

First notice that a formula $\mathsf{EF}\varphi$ is true on a k-weighted modal automaton iff it is true on its *must-projection* that is obtained by removing all the may-transitions that do not have a corresponding must-transition. The reason is that the may-player can, in any may-state, decide to stop the game and hence the formula φ must be satisfied in some configuration reachable via must-transitions only. So in the following we assume that $\dashrightarrow = \longrightarrow$. Since the intervals are assumed singletons, $\mathcal{S}$ is an implementation, and the may-player has no control of the game, i.e. it can be seen as a one-player game for the must-player.

Second, due to Lemma 10 on page 143, we can assume that φ solely consists of simple linear expressions, i.e. expressions of the form $\langle i \rangle \bowtie b$, where $\bowtie \in \{\leq, \geq\}$ and $b \in \mathbb{Z} \cup \{-\infty, \infty\}$.

The translation proceeds as follows. For each state $s \in S$ we add a place p_s to P. To simulate the accumulated weights, we introduce places that store tokens according to the increase and decrease of the weights by performing transitions. Let O_φ be the multiset of linear expressions occurring in the formula φ (each occurrence of any linear expression is included). Each such expression $o \in O_\varphi$ addresses exactly one coordinate



(a) Translation of a 2-weighted transition $(s', (-2, 5), s') \in \longrightarrow$ to a Petri net

(b) Checking a simple linear expression $\psi = \langle i \rangle \leq b$ at occurrence o

Figure 4.: Gadgets used in the reduction

$w(o) \in \{1, \ldots, k\}$. For each $o \in O_\varphi$ with $w(o) = i$ we add two places $p^+_{o,i}$ (a *plus* place) and $p^-_{o,i}$ (a *minus* place) to P. Any increase in coordinate j is simulated by adding tokens to the plus places $p^+_{o,j}$, and any decrease is simulated by adding tokens to the minus places $p^-_{o,j}$. Thus transitions of $\mathcal{S}$ are translated according to the following scheme. For each must-transition $(s, \overline{w}, s') \in \longrightarrow$ in $\mathcal{S}$ we add a transition t to T, and we let $W(p_s, t) = W(t, p_s) = 1$. In order to capture the change of each weight coordinate j we have to add multiplicities, for each $o \in O_\varphi$ such that $w(o) = j$. If $\overline{w}[j] > 0$, then we let $W(t, p^+_{o,j}) = \overline{w}[j]$. If $\overline{w}[j] < 0$, then we let $W(t, p^-_{o,j}) = -\overline{w}[j]$. Fig. 4a shows an example where $\mathcal{S}$ contains one must-transition with weight $(-2, 5)$ and φ comprises two simple linear expressions o and o' addressing coordinates 1 and 2, respectively.

At any moment the Petri net N should be able to enter a checking phase in order to check whether the corresponding configuration in S satisfies φ or not. By induction on the structure of φ we construct a net gadget checking the formula satisfaction. The gadget shown in Fig. 4b shows how to check satisfaction of any occurrence o of simple linear expression the form $\psi = \langle i \rangle \leq b$, which corresponds to checking $M(p^+_{o,i}) \leq M(p^-_{o,i}) + b$. This gadget is activated by putting one token in p^{in}_ψ. Then we add b tokens to $p^-_{o,i}$ when firing transition t_1. Now transition t_2 can be fired a number of times until either $p^+_{o,i}$ or $p^-_{o,i}$ are empty. Firing transition t_4 ensures that $p^-_{o,i}$ for sure can be emptied. It is therefore only possible to empty all places except p^{out}_ψ iff $\langle i \rangle \leq b$ is satisfied. If this is not the case, some tokens in $p^+_{o,i}$ are impossible to remove, since $p^-_{o,i}$ runs out of tokens before $p^+_{o,i}$. The gadget for checking the formula $\langle i \rangle \geq b$ is similar, the only change is the update of the following two multiplicities to arcs: $W(p^-_{o,i}, t_4) = 0$ and $W(p^+_{o,i}, t_4) = 1$.

The conjunction of two formulae, $\psi = \psi_1 \wedge \psi_2$, is checked by checking the two subformulae one at a time by putting their gadgets in series. The gadget for this check is seen in Fig. 5. The gadgets checking ψ_1 and ψ_2 are inserted in the corresponding dashed box. All places can be emptied (except p^{out}_ψ) iff both subformulae are true.

For the disjunction of two formulae, $\psi = \psi_1 \vee \psi_2$, we need to provide two paths in N; one satisfying ψ_1 and one satisfying ψ_2. The gadget is seen in Fig. 6. The gadgets entitled 'Empty ψ' are designed to empty all places associated with ψ, that is all $p^+_{o,i}$

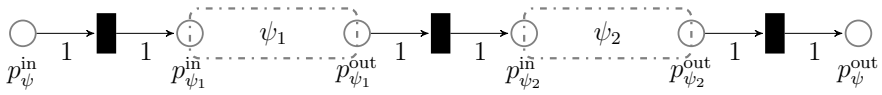


Figure 5.: Checking the formula $\psi = \psi_1 \wedge \psi_2$

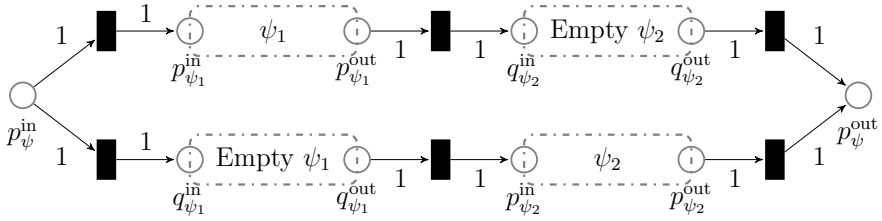


Figure 6.: Checking the formula $\psi = \psi_1 \vee \psi_2$

and $p_{o,i}^-$ with $o \in O_\varphi$. This gadget is seen in Fig. 7. When q' receives a token, repeatedly firing the transitions below q' will provide the existence of an execution where all places have no tokens when passing on the token to q_ψ^{out}. All arcs in Fig. 6 have multiplicity 1.

The whole Petri net N is constructed by adding the gadgets described for simulating any run of $\mathcal{S}$ including the accumulated weights and creating a gadget for checking φ (containing a number of smaller gadgets depending on the structure of φ). In order to connect the checking part to the remaining Petri net (see Fig. 4a), we add for all $s \in S$ a transition t_s to T along with two arcs with multiplicities $W(p_s, t_s) = W(t_s, p_\varphi^{\text{in}}) = 1$. Now from every place p_s in N holding a token it is possible to fire t_s and add a token to p_φ^{in}, from where it is possible to check whether φ is satisfied in s or not.

Let us define now the initial marking as $M_0(p_{s_0}) = 1$, $M(p_{o,i}^+) = \overline{w}_0[i]$ if $\overline{w}_0[i] \geq 0$ and $M(p_{o,i}^-) = -\overline{w}_0[i]$ if $\overline{w}_0[i] < 0$ for all o and $M(p) = 0$ for any other place p. The marking we want to reach is defined as $M_f(p) = 0$ for all $p \in P \setminus \{p_\varphi^{\text{out}}\}$ and $M_f(p_\varphi^{\text{out}}) = 1$. As argued above, the marking M_f is reachable from the initial marking M_0 iff $\mathcal{S} \models \mathsf{EF}\varphi$. This run in N corresponds to simulating $\mathcal{S}$ until reaching the configuration satisfying φ and then entering the checking phase at this moment.

To sum up, we have shown how to reduce the model checking problems for the EF fragment on weighted modal automata with singleton intervals to the decidable reachability problem on Petri nets [May81]. ■

We shall now argue that by restricting the formulae to contain only conjunctions, some fragments of the logic become decidable.

First we show that with singleton intervals (though still allowing modalities) the model checking problem for any $\mathcal{L}$-formula reduces to the same problem for a formula with so-called simple linear expressions. A linear expression is *simple* if it is of the form $\langle i \rangle \bowtie b$, where $\bowtie \in \{\leq, \geq\}$ and $b \in \mathbb{Z}$.

Lemma 10. *Model checking of an $\mathcal{L}$-formula against a weighted modal automaton with singleton intervals is polynomial time reducible to model checking an $\mathcal{L}$-formula where all linear expressions are simple.*

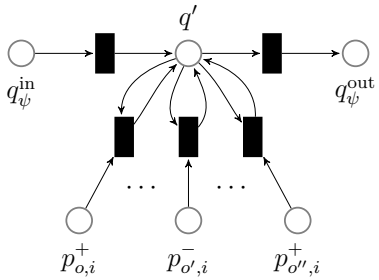


Figure 7.: Gadget for emptying places associated with ψ

Proof. Let $\mathcal{S} = (S, s_0, \overline{w}_0, \dashrightarrow, \longrightarrow)$ be a k-weighted modal automaton with only singleton intervals. Consider any linear expression of the form $e = \langle i_1 \rangle \cdot c_1 + \ldots + \langle i_n \rangle \cdot c_n$, $i_1, \ldots, i_n \in \{1, \ldots, k\}$. Notice that for any modal automaton with singleton intervals, step (ii) in the game can be ignored, since the weights are already uniquely given.

We now construct a $(k+1)$-weighted automaton $\mathcal{T} = (S, s_0, \overline{w}'_0, \dashrightarrow_{\mathcal{T}}, \longrightarrow_{\mathcal{T}})$, where $\overline{w}'_0 = (\overline{w}_0[1], \ldots, \overline{w}_0[k], \overline{w}_0[i_1] \cdot c_1 + \ldots + \overline{w}_0[i_n] \cdot c_n)$. For each $s \overset{\overline{w}}{\dashrightarrow} s'$ in $\dashrightarrow$ we add $s \overset{\overline{v}}{\dashrightarrow} s'$ to $\dashrightarrow_{\mathcal{T}}$, where $\overline{v} = (\overline{w}[1], \ldots, \overline{w}[k], \overline{w}[i_1] \cdot c_1 + \ldots + \overline{w}[i_n] \cdot c_n)$. The set $\longrightarrow_{\mathcal{T}}$ is constructed similarly. Now any linear expression $e \bowtie b$, $\bowtie \in \{<, \leq, =, \neq, \geq, >\}$, $b \in \mathbb{Z}$, is satisfied in $\mathcal{S}$ if and only if the state formula $\langle k+1 \rangle \bowtie b$ is satisfied in $\mathcal{T}$ for $\bowtie \in \{\leq, \geq\}$. The relations $<$, $=$, $\neq$ and $>$ can be modelled using only $\leq$ and $\geq$. For $e \bowtie b$, where $\bowtie \in \{<, >\}$ we instead use $e \leq b - 1$ and $e \geq b + 1$, respectively. For $\bowtie$, where $\bowtie \in \{=, \neq\}$ we use $e \leq b \wedge e \geq b$ and $e < b \wedge e > b$, respectively. ■

We now prove that the disjunction-free EG fragment is decidable, by showing a reduction to the so-called *multiweighted energy games* [FJLS11].

A k-weighted energy game is a four tuple $G = (S_1, S_2, s_0, \longrightarrow)$ where S_1 and S_2 are finite disjoint sets of existential and universal states, respectively, $s_0 \in S_1 \cup S_2$ is the start state and $\longrightarrow \subseteq (S_1 \cup S_2) \times \mathbb{Z}^k \times (S_1 \cup S_2)$ is a finite multiweighted transition relation. Furthermore a k-weighted game is non-blocking, meaning that all states have some outgoing transition.

Configurations, plays and strategies are defined similarly to the same terms on k-weighted modal automata. A configuration is a pair $(s, \overline{w})$, where $s \in S_1 \cup S_2$ and $\overline{w} \in \mathbb{Z}^k$, while a play is an infinite sequence of configurations $(s_0, \overline{w}_0)(s_1, \overline{w}_1) \ldots$ such that $(s_i, \overline{v}_i, s_{i+1}) \in \longrightarrow$ and $\overline{w}_i + \overline{v}_i = \overline{w}_{i+1}$ for all $i \geq 0$. A strategy for the existential player is a mapping σ from each finite prefix of a play $(s_0, \overline{w}_0) \ldots (s_n, \overline{w}_n)$ such that $s_n \in S_1$ to a configuration $(s_{n+1}, \overline{w}_{n+1})$ such that $(s_0, \overline{w}_0) \ldots (s_n, \overline{w}_n)(s_{n+1}, \overline{w}_{n+1})$ is a prefix of a play in G. Given a strategy σ, the set of all plays in G of the form $(s_0, \overline{w}_0)(s_1, \overline{w}_1) \ldots$, where $\overline{w}_0 = \overline{0}$ and $\sigma((s_0, \overline{w}_0) \ldots (s_n, \overline{w}_n)) = (s_{n+1}, \overline{w}_{n+1})$ for all $s_n \in S_1$ is called $plays(\sigma, G)$. Given a k-weighted game G and a $\overline{b} \in \mathbb{N}^k$, the energy game with upper bound asks whether there exists a strategy σ for the existential player such that all $(s_0, \overline{w}_0), (s_1, \overline{w}_1) \ldots \in plays(\sigma, G)$ satisfy $\overline{0} \leq \overline{w}_i \leq \overline{b}$ for all i. If we have only the requirement of $\overline{0} \leq \overline{w}_i$ for all i, we call it an energy game with lower bound.

Theorem 11. *Model checking the disjunction-free* EG *fragment is polynomial time equivalent to deciding the winner of energy games with lower bound.*

Proof. Let $\mathcal{S} = (S, s_0, \overline{w}, \dashrightarrow, \longrightarrow)$ be a k-weighted modal automaton. We will first reduce the model checking problem of EG on $\mathcal{S}$ to an energy game with only lower bounds.

Due to Lemma 14 we can assume that $\mathcal{S}$ is a weighted modal automaton with singleton intervals. This ensures that we can apply Lemma 10 and assume that all linear expressions in φ are simple. First, we notice that any such φ can be rewritten to an equivalent form

$$\varphi = \left(\bigwedge_{i=1}^{k} \langle i \rangle \geq \ell_i \right) \wedge \left(\bigwedge_{i=1}^{k} \langle i \rangle \leq u_i \right) , \tag{1}$$

where $\ell_i, u_i \in \mathbb{Z} \cup \{-\infty, \infty\}$ and $\ell_i = -\infty$ implies $u_i = \infty$. This follows since a coordinate with no upper bound can safely be bounded above by ∞ and a coordinate with no lower bound can safely be bounded below by $-\infty$. In addition a coordinate with an upper bound and no lower bound can be simulated using only a lower bound by multiplying all weights on transitions and the bound in this coordinate by -1. It is clear that there exits a unique ℓ_i giving the largest lower bound and a unique u_i giving the smallest upper bound for each $i \in \{1, \ldots, k\}$ should there be multiple constrains related to the coordinate i.

We shall now reduce the model checking problem for $\mathsf{EG}\varphi$ to a k-weighted energy game. The game $G_{\mathcal{S}} = (S_1, S_2, s_G, \longrightarrow_G)$ is constructed from $\mathcal{S}$ by splitting S into two sets, such that $s_G = s_0$, $S_1 = \{s \in S \mid s \longrightarrow s' \text{ for some } s' \in S\}$ (the existential states) and $S_2 = S \setminus S_1 \cup \{s'\}$ (the universal states). The state s' is added to S_2 in order to capture the fact that any may-transition can be dropped. In any universal state we therefore add a transition to s' with $\overline{0}$ as weight. Furthermore s' has a selfloop also with $\overline{0}$ as weight vector in order to ensure a non-blocking game. Hence we define

$$\longrightarrow_G = \longrightarrow \cup \dashrightarrow \cup \{s \xrightarrow{\overline{0}}_G s' \mid s \in S_2\} \cup \{s' \xrightarrow{\overline{0}}_G s'\}.$$

Now $\mathcal{S}$ satisfies $\mathsf{EG}\varphi$ iff there exists a strategy σ for the existential player in $G_{\mathcal{S}}$ such that any infinite play that proceeds according to σ and starts in s_G with initial weight $(\overline{w}[1] - \ell_1, \ldots, \overline{w}[k] - \ell_k)$ has accumulated weights which are always nonnegative and do not exceed $(u_1 - \ell_1, \ldots, u_k - \ell_k)$. As proved in [FJLS11] it is possible to remove the upper bounds by doubling the number of weights, hence we get an equivalent instance of the lower bound energy game with $2k$ weights.

To conclude the other direction of the proof, we can realize that any k-weighted energy game $G = (S_1, S_2, s_G, \longrightarrow)$ with only lower bounds can be reduced to a modal automaton $\mathcal{S}$ with singleton intervals by turning all transitions $s \xrightarrow{\overline{w}} s'$, where $s \in S_2$ into only may-transitions. Clearly, the existential player wins the energy game with initial weight $\overline{w}_0$ iff $\mathcal{S} \models \mathsf{EG}(\langle 1 \rangle \geq 0 \wedge \ldots \wedge \langle k \rangle \geq 0)$. ∎

Determining the winner of an energy game with only lower bounds is decidable and EXPSPACE-hard [FJLS11]. This yields the following corollary.

Corollary 12. *Model checking the disjunction-free* EG *fragment on multiweighted modal automata is decidable and EXPSPACE-hard.*

Another complexity result is obtained by reducing the model checking problem of the AG fragment to energy games. This time to the universal version, where all runs must maintain the accumulated weights nonnegative at all times, giving us a polynomial time algorithm.

Theorem 13. *Model checking the disjunction-free* AG *fragment is in P.*

Proof. By using the same reductions as in the proof of Theorem 11, we find an equivalence with the universal energy game with only lower bounds. The only modification is that after constructing the game $G_{\mathcal{S}} = (S_1, S_2, s_G, \longrightarrow_G)$ we make another game $G_{\mathcal{S}'} = (\emptyset, S_1 \cup S_2, s_G, \longrightarrow_G)$, such that all states belong to the universal player. Deciding a universal energy game (regardless of the bounds) can be done in polynomial time [BFL+08]. ■

We are now ready to prove the decidability of the full AG fragment.

4.3. Decidability of AG

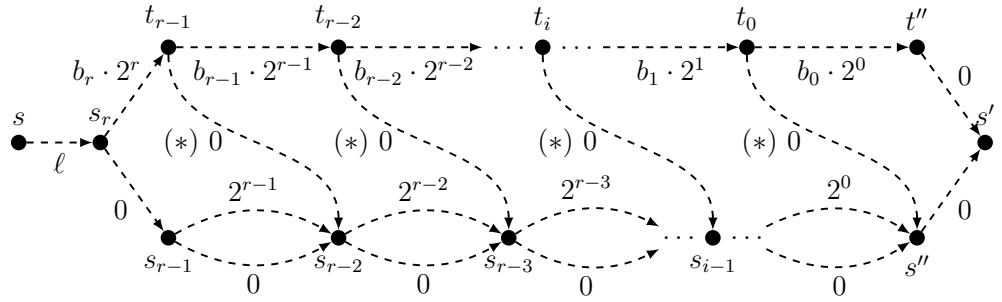
In order to prove that the AG fragment of $\mathcal{L}$ is decidable, we need the following lemma, stating that the weight intervals can be reduced to singleton intervals when model checking a formula in the EG or AG fragment.

Lemma 14. *Let $\mathcal{S} \in \mathbb{M}$ be a k-weighted modal automaton and ψ be a formula from the* EG *or* AG *fragment. We can in polynomial time construct a k-weighted modal automaton $\mathcal{T} \in \mathbb{M}$ with singleton intervals such that*

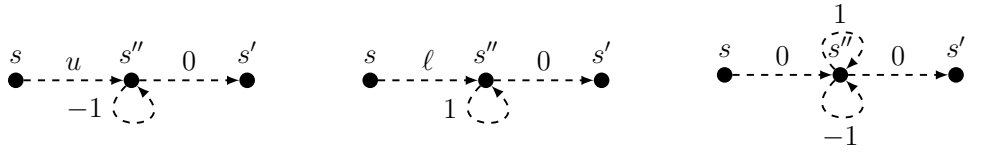
$$\mathcal{S} \models \psi \quad \textit{iff} \quad \mathcal{T} \models \psi.$$

Proof. In order to bypass the exponential blow up that the straightforward reduction of a transition $s \overset{[\ell,u]}{\dashrightarrow} s'$, $\ell, u \in \mathbb{Z}$, (even in the case of $k = 1$) to $u - \ell$ different singleton weighted transitions from s to s' would give, we instead use the following construction. Each transition $s \overset{\overline{W}}{\dashrightarrow} s'$ in $\mathcal{S}$ is translated into a number of transitions in $\mathcal{T}$. For each coordinate i of $\overline{W}$ one of the gadgets depicted in Figure 8 is chosen. Notice that the figure is shown for $k = 1$. Otherwise zeros are put into coordinates different from i. In case $s \overset{\overline{W}}{\longrightarrow} s'$ in $\mathcal{S}$ the transition from s to s'' in each gadget must be a must-transition as well. The k selected gadgets (Figure 8a-8d) for each coordinate are then connected to each other in series in any order. What gadget to use depends on the specific interval, and we thus distinguish between the following three cases.

Both bounds are integers: In this case coordinate i of $\overline{W}$ is bounded by the interval $[\ell, u]$, $\ell, u \in \mathbb{Z}$. We can assume that the size of the interval, $u-\ell$, is written in binary using $r+1$ bits as $b_r b_{r-1} \dots b_1 b_0$ such that $b_r = 1$. The corresponding gadget is given in Figure 8a. We add the lower bound of the interval first ($s \overset{\ell}{\dashrightarrow} s_k$), and using the remaining



(a) Case $\ell, u \in \mathbb{Z}$ (transitions marked with $(*)$, $(t_j, 0, s_{j-1}) \in \dashrightarrow$, are present if $b_j = 1$)



(b) Case $\ell = -\infty$, $u \in \mathbb{Z}$ (c) Case $\ell \in \mathbb{Z}$, $u = \infty$ (d) Case $\ell = -\infty$, $u = \infty$

Figure 8.: Translation of $(s, ([\ell, u]), s') \in \dashrightarrow$ into singleton weighted transitions

construction we let the may-player construct a value $j \in \{0, \ldots, u - \ell\}$, simulating the game semantics of ψ, where the may-player chooses a weight in each weight interval. To see this we observe that taking the path $s_r \overset{b_r \cdot 2^r}{\dashrightarrow} t_{r-1} \overset{b_{r-1} \cdot 2^{r-1}}{\dashrightarrow} t_{r-2} \overset{b_{r-2} \cdot 2^{r-2}}{\dashrightarrow} \ldots \overset{b_1 \cdot 2^1}{\dashrightarrow} t_0 \overset{b_0 \cdot 2^0}{\dashrightarrow} t'' \overset{0}{\dashrightarrow} s'$ (the uppermost path) corresponds to the may-player constructing $u - \ell$ and thus picking the weight u from $\overline{W}$. Taking the lowermost path along transitions with weight 0 corresponds to picking the weight ℓ, while any other path from s to s' builds a weight between ℓ and u. Notice that the may-player cannot construct a weight larger than $u - \ell$, since he only moves to the lowermost part of the figure (where he can construct any number for the remaining bits) if he chooses 0 for any of the bits in $u - \ell$ set to 1.

One bound is an integer, one bound is not: In the second case, where coordinate i of $\overline{W}$ is bounded below by $-\infty$ or above by ∞ (but not both) we use either the gadget in Figure 8b or the gadget in Figure 8c. We start by adding the upper bound (in case the lower bound is $-\infty$) or the lower bound (in case the upper bound is ∞) and then decrease (or increase) the first coordinate by an arbitrary number.

Both bounds are not integers: The last case, where the interval equals the set $\mathbb{Z}$, we use the gadget depicted in Figure 8d in order to decrease or increase the weight arbitrarily.

Now $\mathcal{S} \models \psi$ if and only if $\mathcal{T} \models \psi$. This is true since the may-player cannot choose the weights differently in $\mathcal{T}$ than he could have done in $\mathcal{S}$. The may-player can however choose to stop anywhere inside the gadget, but this cannot break the satisfiability of ψ due to the G quantifier. ■

We are now ready to prove the decidability of the AG fragment.

Theorem 15. *The model checking problem of the* AG *fragment of* $\mathcal{L}$ *is decidable.*

Proof. Let $\mathcal{S} \in \mathbb{M}$ and let $\mathsf{AG}\varphi$ be a formula from the AG fragment. By Lemma 14 we can reduce $\mathcal{S}$ to a weighted modal automaton with only singleton intervals. Furthermore we may consider only the *may-projection* of $\mathcal{S}$ obtained by turning all may-transitions into must-transitions, thus obtaining the most permissible implementation. Now φ must be satisfied in every reachable configuration. Since $\mathcal{S}$ is an implementation, checking $\mathsf{AG}\varphi$ corresponds to checking the negation of $\mathsf{EF}\neg\varphi$ and hence it is decidable by Theorem 9. ■

5. Conclusion

We studied multiweighted modal automata and proposed a fragment of CTL in order to reason about the accumulated weights gathered along the execution of the system, a practically motivated problem where one of its particular instances called energy games has recently become an active research topic. The semantics of the logic is given in terms of two-player games and the definitions were justified by showing that the fragment is both sound and, contrary to the previous attempts, also complete. We believe that a game-semantics is necessary for achieving the completeness of the logic, and the paper takes a first step in this direction. We showed that the logic is in general undecidable, but there are reasonable fragments that are practically interesting and remain decidable.

There are various directions for future works. Clearly, larger fragments of CTL (or generally of the μ-calculus), should be identified for which both soundness and completeness can be obtained. As the model checking problem is in general undecidable, a possible way to attack the problem can be to extend our framework to a three-valued formalism with refinement, like in the spirit of [GP09]. Another direction for future work is to extend the results to a more general setting using lattices, as in [KL07].

Bibliography

[AdAdS+06] B. T. Adler, L. de Alfaro, L. D. da Silva, M. Faella, A. Legay, V. Raman, and P. Roy. Ticc: A Tool for Interface Compatibility and Composition. In *18th International Conference on Computer Aided Verification (CAV'06)*, volume 4144 of *Lecture Notes in Computer Science*, pages 59–62. Springer, 2006.

[AH06] A. Antonik and M. Huth. Efficient Patterns for Model Checking Partial State Spaces in CTL *intersection* LTL. *Electronic Notes in Theoretical Computer Science*, 158:41 – 57, 2006.

[AH09] M. Atig and P. Habermehl. On Yen's Path Logic for Petri Nets. In *3rd International Workshop on Reachability Problems (RP'03)*, volume 5797 of *Lecture Notes in Computer Science*, pages 51–63. Springer-Verlag, 2009.

[AHL+08] A. Antonik, M. Huth, K. G. Larsen, U. Nyman, and A. Wąsowski. 20 Years of Modal and Mixed Specifications. *Bulletin of the EATCS*, (95):94–129, 2008.

[ASW94] H. R. Andersen, C. Stirling, and G. Winskel. A Compositional Proof System for the Modal mu-Calculus. In *9th Annual IEEE Symposium on Logic in Computer Science (LICS'94)*, pages 144–153. IEEE Computer Society, 1994.

[ATP04] R. Alur, S. L. Torre, and G. J. Pappas. Optimal Paths in Weighted Timed Automata. *Theoretical Computer Science*, 318(3):297–322, 2004.

[AVARB+01] Y. Abarbanel-Vinov, N. Aizenbud-Reshef, I. Beer, C. Eisner, D. Geist, T. Heyman, I. Reuveni, E. Rippel, I. Shitsevalov, Y. Wolfsthal, and T. Yatzkar-Haham. On the Effective Deployment of Functional Formal Verification. *Formal Methods in System Design*, 19(1):35–44, 2001.

[BCHK11] U. Boker, K. Chatterjee, T. Henzinger, and O. Kupferman. Temporal Specifications with Accumulative Values. In *26th Annual IEEE Symposium on Logic in Computer Science (LICS'11)*, pages 43–52. IEEE Computer Society, 2011.

[BFH+01] G. Behrmann, A. Fehnker, T. Hune, K. G. Larsen, P. Pettersson, J. Romijn, and F. Vaandrager. Minimum-Cost Reachability for Priced Timed Automata. In *4th International Workshop on Hybrid Systems:*

Computation and Control (HSCC'01), volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer-Verlag, 2001.

[BFJ+11] S. S. Bauer, U. Fahrenberg, L. Juhl, K. G. Larsen, A. Legay, and C. Thrane. Quantitative Refinement for Weighted Modal Transition Systems. In *36th International Symposium on Mathematical Foundations of Computer Science (MFCS'11)*, volume 6907 of *Lecture Notes in Computer Science*, pages 60–71. Springer-Verlag, 2011.

[BFL+08] P. Bouyer, U. Fahrenberg, K. G. Larsen, N. Markey, and J. Srba. Infinite Runs in Weighted Timed Automata with Energy Constraints. In *6th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'08)*, volume 5215 of *Lecture Notes in Computer Science*, pages 33–47. Springer-Verlag, 2008.

[BFLM10] P. Bouyer, U. Fahrenberg, K. G. Larsen, and N. Markey. Timed Automata with Observers under Energy Constraints. In *13th ACM International Conference on Hybrid Systems: Computation and Control (HSCC'10)*, pages 61–70. ACM, 2010.

[BG00] G. Bruns and P. Godefroid. Generalized Model Checking: Reasoning about Partial State Spaces. In *11th International Conference on Concurrency Theory (CONCUR'00)*, volume 1877 of *Lecture Notes in Computer Science*, pages 168–182. Springer-Verlag, 2000.

[BG09] B. Bollig and P. Gastin. Weighted versus Probabilistic Logics. In Diekert and Nowotka [DN09], pages 18–38.

[BJK10] T. Brázdil, P. Jančar, and A. Kučera. Reachability Games on Extended Vector Addition Systems with States. In *37th International Colloquium on Automata, Languages and Programming (ICALP'10), Part II*, volume 6199 of *Lecture Notes in Computer Science*, pages 478–489. Springer-Verlag, 2010.

[BK84] J. A. Bergstra and J. W. Klop. Process Algebra for Synchronous Communication. *Information and Control*, 60(1-3):109–137, 1984.

[BK90] J. C. M. Baeten and J. W. Klop, editors. *Theories of Concurrency: Unification and Extension (CONCUR'90)*, volume 458 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[BK08] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

[BK11] N. Beneš and J. Kretínský. Process Algebra for Modal Transition Systemses. In *6th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS'10), Selected Papers*, volume 16 of *OASIcs*, pages 9–18. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2011.

[BKL+11] N. Beneš, J. Kretínský, K. G. Larsen, M. H. Møller, and J. Srba. Parametric Modal Transition Systems. In *9th International Symposium on Automated Technology for Verification and Analysis, (ATVA'11)*, volume 6996 of *Lecture Notes in Computer Science*, pages 275–289. Springer-Verlag, 2011.

[BKLS09a] N. Beneš, J. Křetínský, K. G. Larsen, and J. Srba. Checking Thorough Refinement on Modal Transition Systems Is EXPTIME-Complete. In *6th International Colloquium on Theoretical Aspects of Computing (ICTAC'09)*, volume 5684 of *Lecture Notes in Computer Science*, pages 112–126. Springer-Verlag, 2009.

[BKLS09b] N. Beneš, J. Křetínský, K. G. Larsen, and J. Srba. On Determinism in Modal Transition Systems. *Theoretical Computer Science*, 410(41):4026–4043, 2009.

[BL92] G. Boudol and K. G. Larsen. Graphical versus logical specifications. *Theoretical Computer Science*, 106(1):3–20, 1992.

[BLL+11] S. Bauer, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski. A Modal Specification Theory for Components with Data. In *8th International Symposium on Formal Aspects of Component Software (FACS'11)*, 2011. To appear in LNCS.

[BLPR09] N. Bertrand, A. Legay, S. Pinchinat, and J.-B. Raclet. A Compositional Approach on Modal Specifications for Timed Systems. In *Formal Methods and Software Engineering, 11th International Conference on Formal Engineering Methods (ICFEM'09)*, volume 5885 of *Lecture Notes in Computer Science*, pages 679–697. Springer-Verlag, 2009.

[BMSH10] S. S. Bauer, P. Mayer, A. Schroeder, and R. Hennicker. On Weak Modal Compatibility, Refinement, and the MIO Workbench. In *16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'10)*, volume 6015 of *Lecture Notes in Computer Science*, pages 175–189. Springer-Verlag, 2010.

[BPR09] N. Bertrand, S. Pinchinat, and J.-B. Raclet. Refinement and Consistency of Timed Modal Specifications. In *3rd International Conference on Language and Automata Theory and Applications (LATA'09)*, volume 5457 of *Lecture Notes in Computer Science*, pages 152–163. Springer-Verlag, 2009.

[BvK10] N. Beneš, I. Černá, and J. Křetínský. Disjunctive Modal Transition Systems and Generalized LTL Model Checking. Technical Report FIMU-RS-2010-12, Faculty of Informatics, Masaryk University, 2010.

[CAP+98] W. Chan, R. J. Anderson, B. Paul, S. Burns, F. Modugno, D. Notkin, and J. D. Reese. Model Checking Large Software Specifications. *IEEE Transactions on Software Engineering*, 24:498–520, 1998.

[CD10] K. Chatterjee and L. Doyen. Energy Parity Games. In *37th International Colloquium on Automata, Languages and Programming (ICALP'10), Part II*, volume 6199 of *Lecture Notes in Computer Science*, pages 599–610. Springer-Verlag, 2010.

[CdAHS03] A. Chakrabarti, L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Resource Interfaces. In *3rd International Conference on Embedded Software (EMSOFT'03)*, volume 2855 of *Lecture Notes in Computer Science*, pages 117–133. Springer-Verlag, 2003.

[CDHR10] K. Chatterjee, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Generalized Mean-payoff and Energy Games. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'10)*, volume 8 of *LIPIcs*, pages 505–516. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2010.

[CDL+10] B. Caillaud, B. Delahaye, K. G. Larsen, A. Legay, M. L. Pedersen, and A. Wasowski. Compositional Design Methodology with Constraint Markov Chains. In *7th International Conference on Quantitative Evaluation of SysTems (QEST'10)*, pages 123–132. IEEE Computer Society, 2010.

[CE82] E. M. Clarke and E. A. Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Logics of Programs, Workshop 1981*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1982.

[CGL93] K. Cerans, J. C. Godskesen, and K. G. Larsen. Timed Modal Specification - Theory and Tools. In *5th International Conference on Computer Aided Verification,(CAV'93)*, volume 697 of *Lecture Notes in Computer Science*, pages 253–267. Springer-Verlag, 1993.

[Cha10] J. Chaloupka. Z-Reachability Problem for Games on 2-Dimensional Vector Addition Systems with States Is in P. In *4th International Workshop on Reachability Problems (RP'10)*, volume 6227 of *Lecture Notes in Computer Science*, pages 104–119. Springer-Verlag, 2010.

[Cir79] R. Cirillo. *The Economics of Vilfredo Pareto.* Frank Cass Publishers, 1979.

[COM11] COMBEST, 2008–2011. http://www.combest.eu.com.

[dAdSF+05] L. de Alfaro, L. D. da Silva, M. Faella, A. Legay, P. Roy, and M. Sorea. Sociable Interfaces. In *5th International Conference on Frontiers of Combining Systems (FROCOS'05)*, volume 3717 of *Lecture Notes in Computer Science*, pages 81–105. Springer, 2005.

[dAF03] L. de Alfaro and M. Faella. Information Flow in Concurrent Games. In *30th International Colloquium on Automata, Languages and Programming (ICALP'03)*, volume 2719 of *Lecture Notes in Computer Science*, pages 1038–1053. Springer-Verlag, 2003.

[dAFS04] L. de Alfaro, M. Faella, and M. Stoelinga. Linear and Branching Metrics for Quantitative Transition Systems. In *31st International Colloquium on Automata, Languages and Programming (ICALP'04)*, volume 3142 of *Lecture Notes in Computer Science*, pages 97–109. Springer-Verlag, 2004.

[dAFS09] L. de Alfaro, M. Faella, and M. Stoelinga. Linear and Branching System Metrics. *IEEE Transactions on Software Engineering*, 35(2):258–273, 2009.

[dAH01] L. de Alfaro and T. A. Henzinger. Interface Automata. In *8th ESEC and 9th ACM SIGSOFT FSE*, ESEC/FSE-9, pages 109–120. ACM, 2001.

[dAH05] L. de Alfaro and T. A. Henzinger. Interface-Based Design. In *Engineering Theories of Software-intensive Systems*, volume 195 of *NATO Science Series: Mathematics, Physics, and Chemistry*, pages 83–104. Springer-Verlag, 2005.

[dAHS02] L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Timed Interfaces. In *2nd International Conference on Embedded Software (EMSOFT'02)*, volume 2491 of *Lecture Notes in Computer Science*, pages 108–122. Springer-Verlag, 2002.

[dAM01] L. de Alfaro and R. Majumdar. Quantitative Solution of Omega-Regular Games. In *33th Annual ACM Symposium on Theory of Computing (STOC'01)*, pages 675–683. ACM, 2001.

[DDG+10] A. Degorre, L. Doyen, R. Gentilini, J.-F. Raskin, and S. Torunczyk. Energy and Mean-Payoff Games with Imperfect Information. In *24th International Workshop on Computer Science Logic (CSL'10)*, volume 6247 of *Lecture Notes in Computer Science*, pages 260–274. Springer-Verlag, 2010.

[DG07] M. Droste and P. Gastin. Weighted Automata and Weighted Logics. *Theoretical Computer Science*, 380(1-2):69–86, 2007.

[DGJP04] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for Labelled Markov Processes. *Theoretical Computer Science*, 318(3):323–354, 2004.

[DKL+11] B. Delahaye, J.-P. Katoen, K. G. Larsen, A. Legay, M. L. Pedersen, F. Sher, and A. Wasowski. Abstract Probabilistic Automata. In *12th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'11)*, volume 6538 of *Lecture Notes in Computer Science*, pages 324–339. Springer-Verlag, 2011.

[DKV09] M. Droste, W. Kuich, and H. Vogler, editors. *Handbook of Weighted Automata*. Springer-Verlag, 1 edition, 2009.

[DLL+10] A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski. Timed I/O Automata: A Complete Specification Theory for Real-time Systems. In *13th ACM International Conference on Hybrid Systems: Computation and Control (HSCC'10)*, pages 91–100. ACM, 2010.

[DLLW10] B. Delahaye, K. G. Larsen, A. Legay, and A. Wasowski. On Greatest Lower Bound of Modal Transition Systems. Technical report, INRIA, 2010.

[DM10] M. Droste and I. Meinecke. Describing Average- and Longtime-Behavior by Weighted MSO Logics. In *35th International Symposium on Mathematical Foundations of Computer Science (MFCS'10)*, volume 6281 of *Lecture Notes in Computer Science*, pages 537–548. Springer-Verlag, 2010.

[DN09] V. Diekert and D. Nowotka, editors. *13th International Conference on Developments in Language Theory (DLT'09)*, volume 5583 of *Lecture Notes in Computer Science*. Springer-Verlag, 2009.

[EC80] E. A. Emerson and E. M. Clarke. Characterizing Correctness Properties of Parallel Programs Using Fixpoints. In *7th International Colloquium on Automata, Languages and Programming (ICALP'80)*, volume 85 of *Lecture Notes in Computer Science*, pages 169–181. Springer-Verlag, 1980.

[EH86] E. A. Emerson and J. Y. Halpern. "Sometimes" and "Not Never" Revisited: On Branching versus Linear Time Temporal Logic. *Journal of the ACM*, 33:151–178, 1986.

[Eil74] S. Eilenberg. *Automata, Languages, and Machines*. Academic Press, Inc., 1974.

[EM79] A. Ehrenfeucht and J. Mycielski. Positional Strategies for Mean Payoff Games. *International Journal of Game Theory*, 8(2):109–113, 1979.

[Ern05] J. Ernits. Memory Arbiter Synthesis and Verification for a Radar Memory Interface Card. *Nordic Journal of Computing*, 12:68–88, 2005.

[Esp98] J. Esparza. Decidability and Complexity of Petri Net Problems — An Introduction. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 374–428. Springer-Verlag, 1998.

[FJLS11] U. Fahrenberg, L. Juhl, K. G. Larsen, and J. Srba. Energy Games in Multiweighted Automata. In *8th International Colloquium on Theoretical Aspects of Computing (ICTAC'11)*, volume 6916 of *Lecture Notes in Computer Science*, pages 95–115. Springer-Verlag, 2011.

[FLT10] U. Fahrenberg, K. Larsen, and C. Thrane. A Quantitative Characterization of Weighted Kripke Structures in Temporal Logic. *Computing and Informatics*, 29(6+):1311–1324, 2010.

[FLT11] U. Fahrenberg, A. Legay, and C. Thrane. The Quantitative Linear-Time–Branching-Time Spectrum. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'11)*, volume 13 of *LIPIcs*, pages 103–114. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2011.

[FP07] G. Feuillade and S. Pinchinat. Modal Specifications for the Control Theory of Discrete Event Systems. *Discrete Event Dynamic Systems*, 17(2):211–232, 2007.

[FS08] H. Fecher and H. Schmidt. Comparing Disjunctive Modal Transition Systems with an One-Selecting Variant. *Journal of Logic and Algebraic Programming*, 77(1-2):20–39, 2008.

[FTL11] U. Fahrenberg, C. Thrane, and K. G. Larsen. Distances for Weighted Transition Systems: Games and Properties. In *9th Workshop on Quantitative Aspects of Programming Languages (QAPL'11)*, volume 57 of *EPTCS*, pages 134–147, 2011.

[FUB06] D. Fischbein, S. Uchitel, and V. Braberman. A Foundation for Behavioural Conformance in Software Product Line Architectures. In *ISSTA 2006 Workshop on Role of Software Architecture for Testing and Analysis (ROSATEA'06)*, pages 39–48. ACM, 2006.

[GC05] A. Gurfinkel and M. Chechik. How Thorough Is Thorough Enough? In *13th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'05)*, volume 3725 of *Lecture Notes in Computer Science*, pages 65–80. Springer-Verlag, 2005.

[GHJ01] P. Godefroid, M. Huth, and R. Jagadeesan. Abstraction-Based Model Checking Using Modal Transition Systems. In *12th International Conference on Concurrency Theory (CONCUR'01)*, volume 2154 of *Lecture Notes in Computer Science*, pages 426–440. Springer-Verlag, 2001.

[GLS08a] A. Gruler, M. Leucker, and K. D. Scheidemann. Calculating and Modeling Common Parts of Software Product Lines. In *12th International Conference on Software Product Lines (SPLC'08)*, pages 203–212. IEEE Computer Society, 2008.

[GLS08b] A. Gruler, M. Leucker, and K. D. Scheidemann. Modeling and Model Checking Software Product Lines. In *10th IFIP WG 6.1 International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'08)*, volume 5051 of *Lecture Notes in Computer Science*, pages 113–131. Springer-Verlag, 2008.

[Gor79] M. Gordon. *The Denotational Description of Programming Languages*. Springer-Verlag, 1979.

[GP09] P. Godefroid and N. Piterman. LTL Generalized Model Checking Revisited. In Jones and Müller-Olm [JMO09], pages 89–104.

[GS86] S. Graf and J. Sifakis. A Logic for the Specification and Proof of Regular Controllable Processes of CCS. *Acta Informatica*, 23:507–527, 1986.

[HJS01] M. Huth, R. Jagadeesan, and D. A. Schmidt. Modal Transition Systems: A Foundation for Three-Valued Program Analysis. In *10th European Symposium on Progamming (ESOP'01)*, volume 2028 of *Lecture Notes in Computer Science*, pages 155–169. Springer-Verlag, 2001.

[HL89] H. Hüttel and K. G. Larsen. The Use of Static Constructs in A Modal Process Logic. In *Symposium on Logical Foundations of Computer Science: Logic at Botik '89*, volume 363 of *Lecture Notes in Computer Science*, pages 163–180. Springer-Verlag, 1989.

[HLP01] K. Havelund, M. Lowry, and J. Penix. Formal Analysis of a Space-Craft Controller Using SPIN. *IEEE Transactions on Software Engineering*, 27(8):749–765, 2001.

[HM85] M. Hennessy and R. Milner. Algebraic Laws for Nondeterminism and Concurrency. *Journal of the ACM*, 32(1):137–161, 1985.

[HMP05] T. A. Henzinger, R. Majumdar, and V. S. Prabhu. Quantifying Similarities Between Timed Systems. In *3rd International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'05)*, volume 3829 of *Lecture Notes in Computer Science*, pages 226–241. Springer-Verlag, 2005.

[Hoa69] C. A. R. Hoare. An Axiomatic Basis for Computer Programming. *Communications of the ACM*, 12:576–580, 1969.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[Hol89] S. Holmström. A Refinement Calculus for Specifications in Hennessy-Milner Logic with Recursion. *Formal Aspects of Computing*, 1:242–272, 1989.

[HP85] D. Harel and A. Pnueli. On the Development of Reactive Systems. In *Logics and Models of Concurrent Systems*, volume 13 of *Nato Asi Series F*, pages 477–498. Springer-Verlag, 1985.

[HS06] T. A. Henzinger and J. Sifakis. The Embedded Systems Design Challenge. In *14th International Symposium on Formal Methods (FM'06)*, volume 4085 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 2006.

[HS07] T. A. Henzinger and J. Sifakis. The Discipline of Embedded Systems Design. *IEEE Computer*, 40(10):32–40, 2007.

[JLS12] L. Juhl, K. G. Larsen, and J. Srba. Modal Transition Systems with Weight Intervals. *Journal of Logic and Algebraic Programming*, 81(4):408–421, 2012.

[JMO09] N. D. Jones and M. Müller-Olm, editors. *10th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'09)*, volume 5403 of *Lecture Notes in Computer Science*. Springer-Verlag, 2009.

[JRLD07] J. J. Jessen, J. I. Rasmussen, K. G. Larsen, and A. David. Guided Controller Synthesis for Climate Controller Using UPPAAL TIGA. In *5th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'07)*, volume 4763 of *Lecture Notes in Computer Science*, pages 227–240. Springer-Verlag, 2007.

[JS90] C.-C. Jou and S. A. Smolka. Equivalences, Congruences, and Complete Axiomatizations for Probabilistic Processes. In Baeten and Klop [BK90], pages 367–383.

[Kel76] R. M. Keller. Formal Verification of Parallel Programs. *Communications of the ACM*, 19:371–384, 1976.

[KKN09] J.-P. Katoen, D. Klink, and M. R. Neuhäußer. Compositional Abstraction for Stochastic Systems. In *7th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'09)*, volume 5813 of *Lecture Notes in Computer Science*, pages 195–211. Springer-Verlag, 2009.

[KL07] O. Kupferman and Y. Lustig. Lattice Automata. In *8th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'07)*, volume 4349 of *Lecture Notes in Computer Science*, pages 199–213. Springer-Verlag, 2007.

[Koz83] D. Kozen. Results on the Propositional mu-Calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[KS88] S. Kosaraju and G. Sullivan. Detecting Cycles in Dynamic Graphs in Polynomial Time. In *20th Annual ACM Symposium on Theory of Computing (STOC'88)*, pages 398–406. ACM, 1988.

[KS90] P. Kanellakis and S. Smolka. CCS Expressions, Finite State Processes, and Three Problems of Equivalence. *Information and Computation*, 86(1):43–68, 1990.

[KS08] A. Kučera and O. Stražovský. On the Controller Synthesis for Finite-State Markov Decision Processes. *Fundamenta Informaticae*, 82(1-2):141–153, 2008.

[Lar85] K. G. Larsen. A Context Dependent Equivalence between Processes. In *12th Colloquium on Automata, Languages and Programming (ICALP'85)*, volume 194 of *Lecture Notes in Computer Science*, pages 373–382. Springer-Verlag, 1985.

[Lar87] K. G. Larsen. A Context Dependent Equivalence Between Processes. *Theoretical Computer Science*, 49:184–215, 1987.

[Lar89] K. G. Larsen. Modal Specifications. In Sifakis [Sif90], pages 232–246.

[LLM05] A. Lluch-Lafuente and U. Montanari. Quantitative μ-calculus and CTL defined over constraint semirings. *Theoretical Computer Science*, 346(1):135–160, 2005.

[LLW05] K. G. Larsen, U. Larsen, and A. Wasowski. Color-Blind Specifications for Transformations of Reactive Synchronous Programs. In *8th International Conference on Fundamental Approaches to Software Engineering (FASE'05)*, volume 3442 of *Lecture Notes in Computer Science*, pages 160–174. Springer-Verlag, 2005.

[LM87] K. G. Larsen and R. Milner. Verifying a Protocol Using Relativized Bisimulation. In *14th International Colloquium on Automata, Languages and Programming (ICALP'87)*, volume 267 of *Lecture Notes in Computer Science*, pages 126–135. Springer-Verlag, 1987.

[LM92] K. G. Larsen and R. Milner. A Compositional Protocol Verification Using Relativized Bisimulation. *Information and Computation*, 99(1):80–108, 1992.

[LMN05] K. G. Larsen, M. Mikucionis, and B. Nielsen. Online Testing of Real-time Systems Using UPPAAL. In *4th International Workshop on Formal Approaches to Software Testing (FATES'04), Revised Selected Papers*, volume 3395 of *Lecture Notes in Computer Science*, pages 79–94. Springer-Verlag, 2005.

[LNW07a] K. G. Larsen, U. Nyman, and A. Wasowski. On Modal Refinement and Consistency. In *18th International Conference on Concurrency Theory (CONCUR'07)*, volume 4703 of *Lecture Notes in Computer Science*, pages 105–119. Springer-Verlag, 2007.

[LNW07b] K. G. Larsen, U. Nyman, and A. Wasowski. Modal I/O Automata for Interface and Product Line Theories. In *Programming Languages and Systems, 16th European Symposium on Programming (ESOP'07)*, volume 4421 of *Lecture Notes in Computer Science*, pages 64–79. Springer-Verlag, 2007.

[Low96] G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. *Software - Concepts and Tools*, 17(3):93–102, 1996.

[LT88a] K. G. Larsen and B. Thomsen. A Modal Process Logic. In *3rd Annual Symposium on Logic in Computer Science (LICS'88)*, pages 203–210. IEEE Computer Society, 1988.

[LT88b] K. G. Larsen and B. Thomsen. Compositional Proofs by Partial Specification of Processes. In *Mathematical Foundations of Computer Science (MFCS'88)*, volume 324 of *Lecture Notes in Computer Science*, pages 414–423. Springer-Verlag, 1988.

[LX90] K. G. Larsen and L. Xinxin. Equation Solving Using Modal Transition Systems. In *5th Annual IEEE Symposium on Logic in Computer Science (LICS'90)*, pages 108–117. IEEE Computer Society, 1990.

[LX91] K. G. Larsen and L. Xinxin. Compositionality Through an Operational Semantics of Contexts. *Journal of Logic and Computation*, 1(6):761–795, 1991.

[May81] E. Mayr. An Algorithm for the General Petri Net Reachability Problem. In *13th Annual ACM Symposium on Theory of Computing (STOC'81)*, pages 238–246. ACM, 1981.

[Mei09] I. Meinecke. A Weighted μ-Calculus on Words. In Diekert and Nowotka [DN09], pages 384–395.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

[Mil83] R. Milner. Calculi for Synchrony and Asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

[Min67] M. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.

[MKB08] R. Meolic, T. Kapus, and Z. Brezocnik. ACTLW - An Action-Based Computation Tree Logic with Unless Operator. *Information Sciences*, 178(6):1542–1557, 2008.

[MP73] J. Maynard Smith and G. R. Price. The Logic of Animal Conflict. *Nature*, 246(5427):15–18, 1973.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.

[MS07] P. B. Miltersen and T. B. Sørensen. A Near-Optimal Strategy for a Heads-Up No-Limit Texas Hold'em Poker Tournament. In *6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'07)*, pages 191:1–191:8. IFAAMAS, 2007.

[Mye79] G. J. Myers. *The Art of Software Testing*. John Wiley & Sons, Inc., 1979.

[NNN08] S. Nanz, F. Nielson, and H. R. Nielson. Modal Abstractions of Concurrent Behaviour. In *15th International Symposium on Static Analysis (SAS'08)*, volume 5079 of *Lecture Notes in Computer Science*, pages 159–173. Springer-Verlag, 2008.

[NV90] R. D. Nicola and F. W. Vaandrager. Action versus State based Logics for Transition Systems. In *Semantics of Systems of Concurrent Processes*, volume 469 of *Lecture Notes in Computer Science*, pages 407–419. Springer-Verlag, 1990.

[Par71] V. Pareto. *Manual of Political Economy*. Augustus M. Kelley, 1971. Translated by Ann S. Schwier.

[Par81] D. M. R. Park. Concurrency and Automata on Infinite Sequences. In *5th GI-Conference on Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.

[Pel01] D. A. Peled. *Software Reliability Methods*. Springer New York, Inc., 2001.

[Plo81] G. Plotkin. A Structural Approach to Operational Semantics. FN 19, DAIMI, 1981. Computer Science Department, Aarhus University, Denmark.

[Pnu77] A. Pnueli. The Temporal Logic of Programs. In *18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Computer Society, 1977.

[PS00] B. C. Pierce and D. Sangiorgi. Behavioral Equivalence in the Polymorphic Pi-Calculus. *Journal of the ACM*, 47:531–584, 2000.

[PT87] R. Paige and R. Tarjan. Three Partition Refinement Algorithms. *SIAM Journal of Computing*, 16(6):973–989, 1987.

[QS82] J.-P. Queille and J. Sifakis. Specification and Verification of Concurrent Systems in CESAR. In *5th Colloquium on International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer-Verlag, 1982.

[Rac08] J.-B. Raclet. Residual for Component Specifications. *Electronic Notes in Theoretical Computer Science*, 215:93–110, 2008.

[RBB+09] J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, and R. Passerone. Why Are Modalities Good for Interface Theories? In *9th International Conference on Application of Concurrency to System Design (ACSD'09)*, pages 119–127. IEEE Computer Society, 2009.

[SAH+00] J. Staunstrup, H. R. Andersen, H. Hulgaard, J. Lind-Nielsen, K. G. Larsen, G. Behrmann, K. Kristoffersen, A. Skou, H. Leerberg, and N. B. Theilgaard. Practical Verification of Embedded Software. *IEEE Computer*, 33:68–75, 2000.

[Sak09] J. Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

[Sch61] M. P. Schützenberger. On the Definition of a Family of Automata. *Information and Control*, 4(2-3):245–270, 1961.

[Sif90] J. Sifakis, editor. *International Workshop on Automatic Verification Methods for Finite State Systems 1989*, volume 407 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[SPE10] SPEEDS, 2006–2010. http://www.speeds.eu.com.

[Sti87] C. Stirling. Modal Logics for Communicating Systems. *Theoretical Computer Science*, 49:311–347, 1987.

[Sto77] J. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Progamming Language Theory*. The MIT Press, 1977.

[SW89] C. Stirling and D. Walker. CCS, Liveness, and Local Model Checking in the Linear Time Mu-Calculus. In Sifakis [Sif90], pages 166–178.

[TFL10] C. Thrane, U. Fahrenberg, and K. G. Larsen. Quantitative Analysis of Weighted Transition Systems. *Journal of Logic and Algebraic Programming*, 79(7):689–703, 2010.

[Thr11] C. Thrane. *Models and Analysis for Reactive Systems*. PhD thesis, Aalborg University, 2011.

[TWC01] J. Tretmans, K. Wijbrans, and M. R. V. Chaudron. Software Engineering with Formal Methods: The Development of a Storm Surge Barrier Control System Revisiting Seven Myths of Formal Methods. *Formal Methods in System Design*, 19(2):195–215, 2001.

[UC04] S. Uchitel and M. Chechik. Merging Partial Behavioural Models. In *12th ACM SIGSOFT International Symposium on Foundations of Software Engineerings (FSE'04)*, pages 43–52. ACM, 2004.

[vG90] R. J. van Glabbeek. The Linear Time-Branching Time Spectrum (Extended Abstract). In Baeten and Klop [BK90], pages 278–297.

[VJ84] R. Valk and M. Jantzen. The residue of vector sets with applications to decidability problems in Petri nets. In *Advances in Petri Nets 1984*, volume 188 of *Lecture Notes in Computer Science*, pages 234–258. Springer-Verlag, 1984.

[vNM44] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[WGC09] O. Wei, A. Gurfinkel, and M. Chechik. Mixed Transition Systems Revisited. In Jones and Müller-Olm [JMO09], pages 349–365.

[Yen92] H. Yen. A Unified Approach for Deciding the Existence of Certain Petri Net Paths. *Information and Computation*, 96(1):119–137, 1992.

[ZP96] U. Zwick and M. Paterson. The Complexity of Mean Payoff Games on Graphs. *Theoretical Computer Science*, 158(1&2):343–359, 1996.